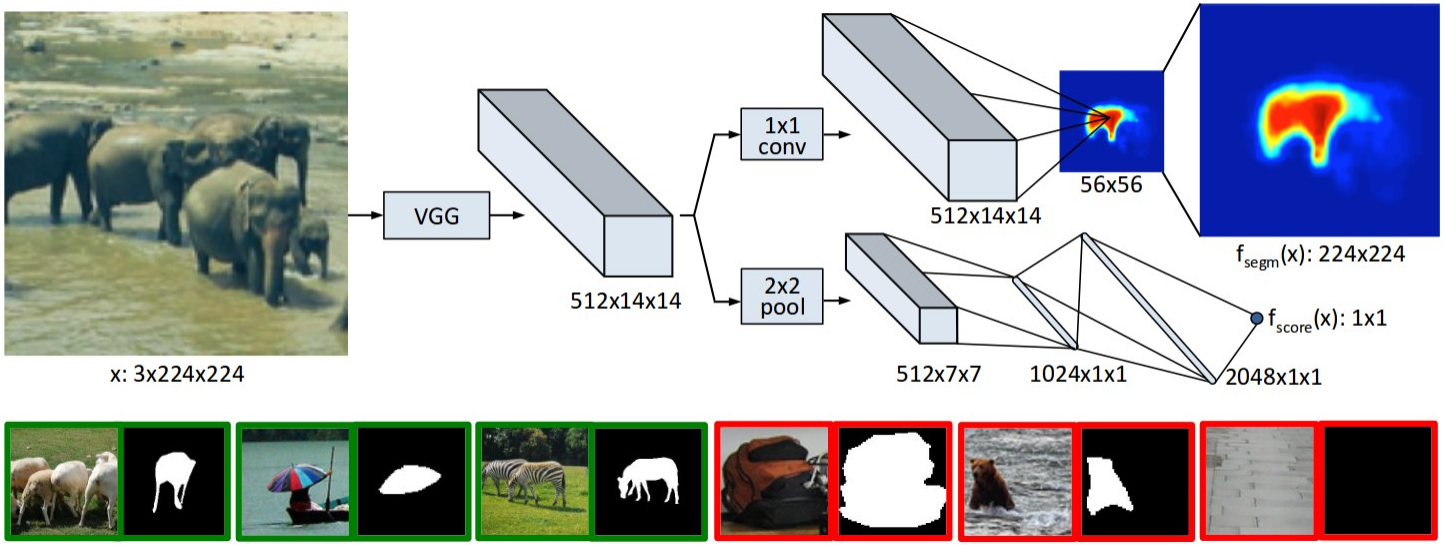
Unsupervised Learning

Yann LeCun
Facebook AI Research &
Center for Data Science, NYU
yann@cs.nyu.edu
http://yann.lecun.com

What Supervised Learning Can Do Today

- [Pinheiro, Collobert, Dollar ICCV 2015]
  - ConvNet produces object masks

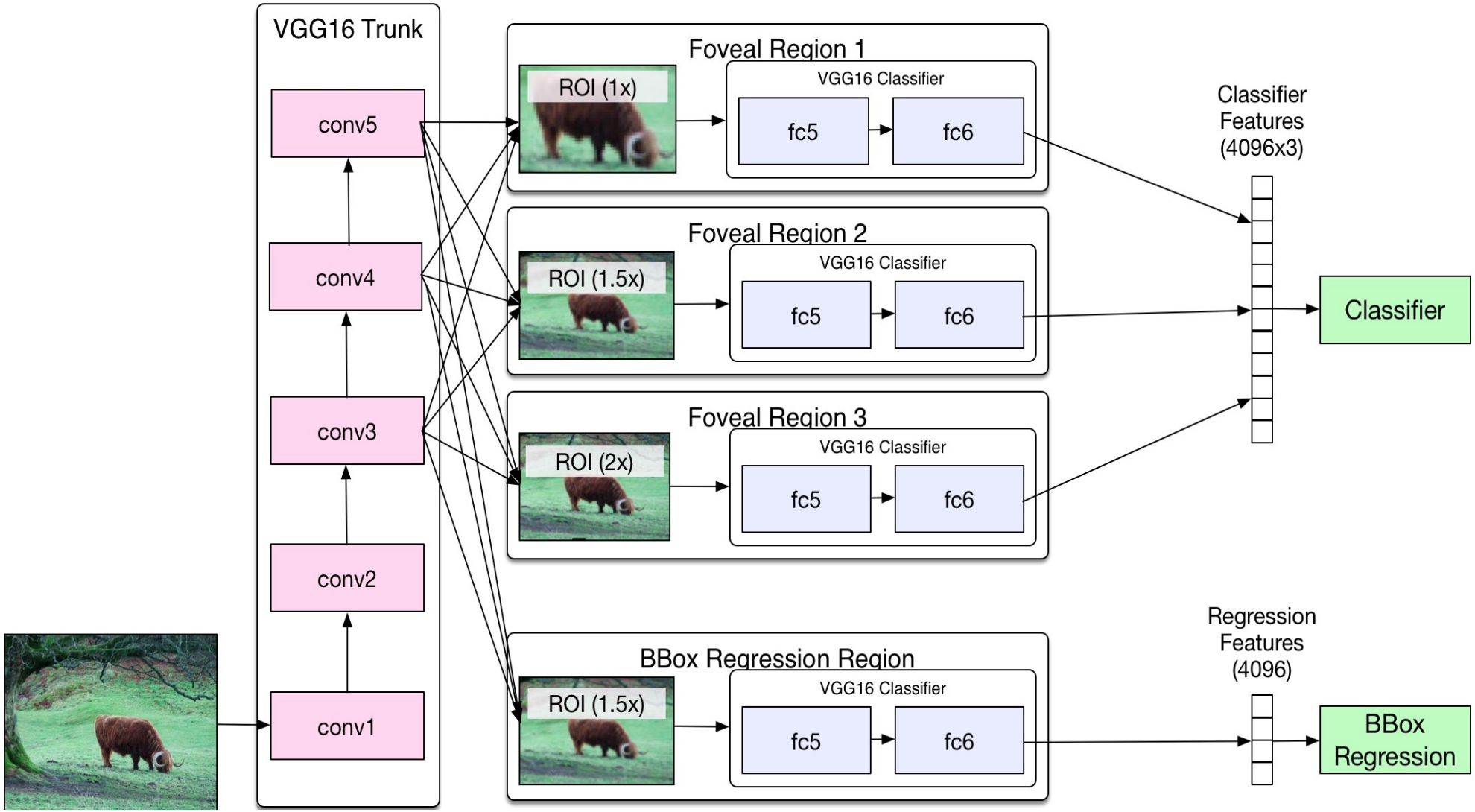


DeepMask++ Proposals

- FAIR COCO Team

Recognition Pipeline

FAIR COCO Team

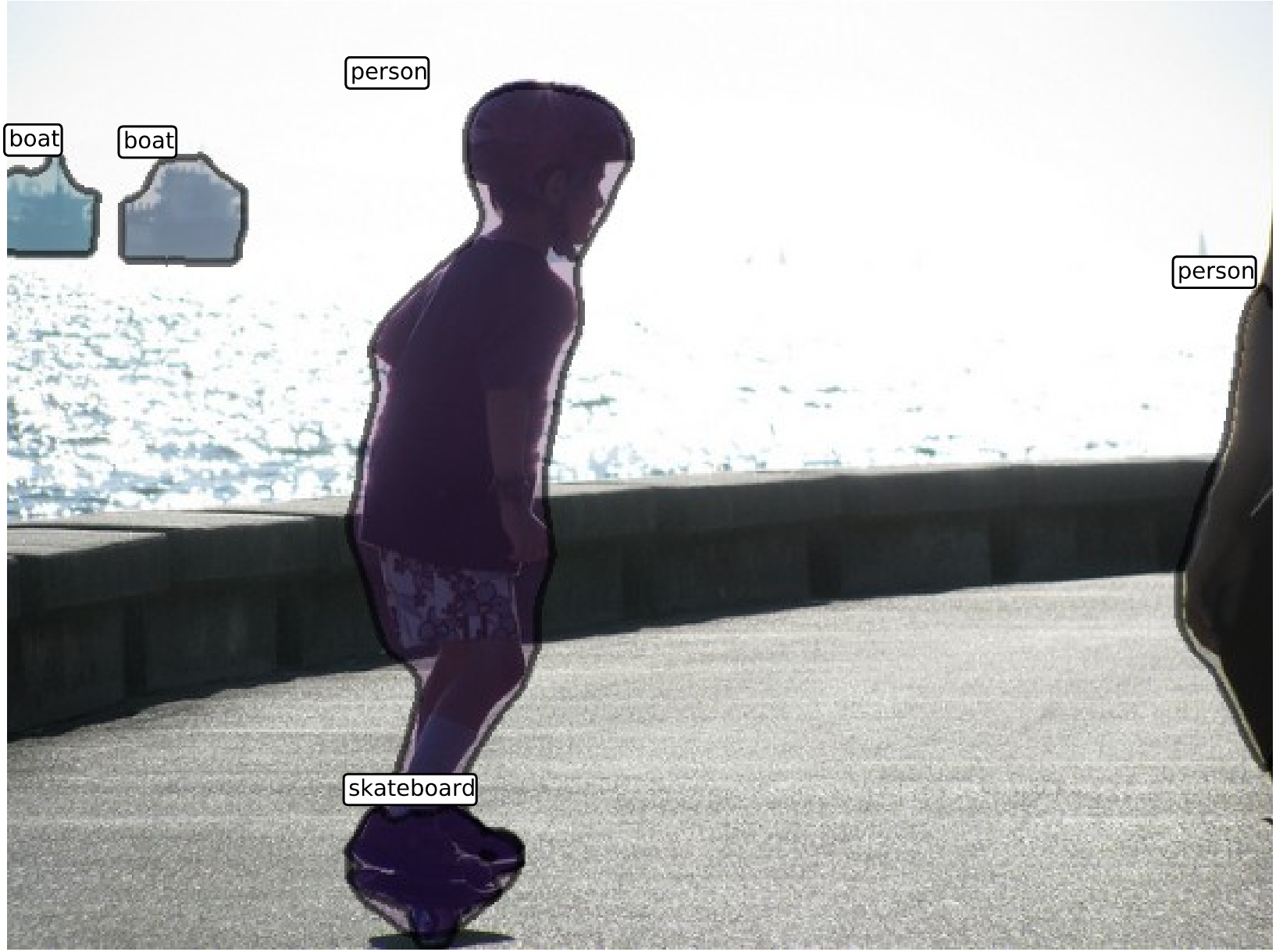


Training

- 2.5 days on 8x4 Kepler GPUs with Elastic Avergaing Stochastic Gradient Descent (EASGD [Zhang, Choromanska, LeCun NIPS 2015]

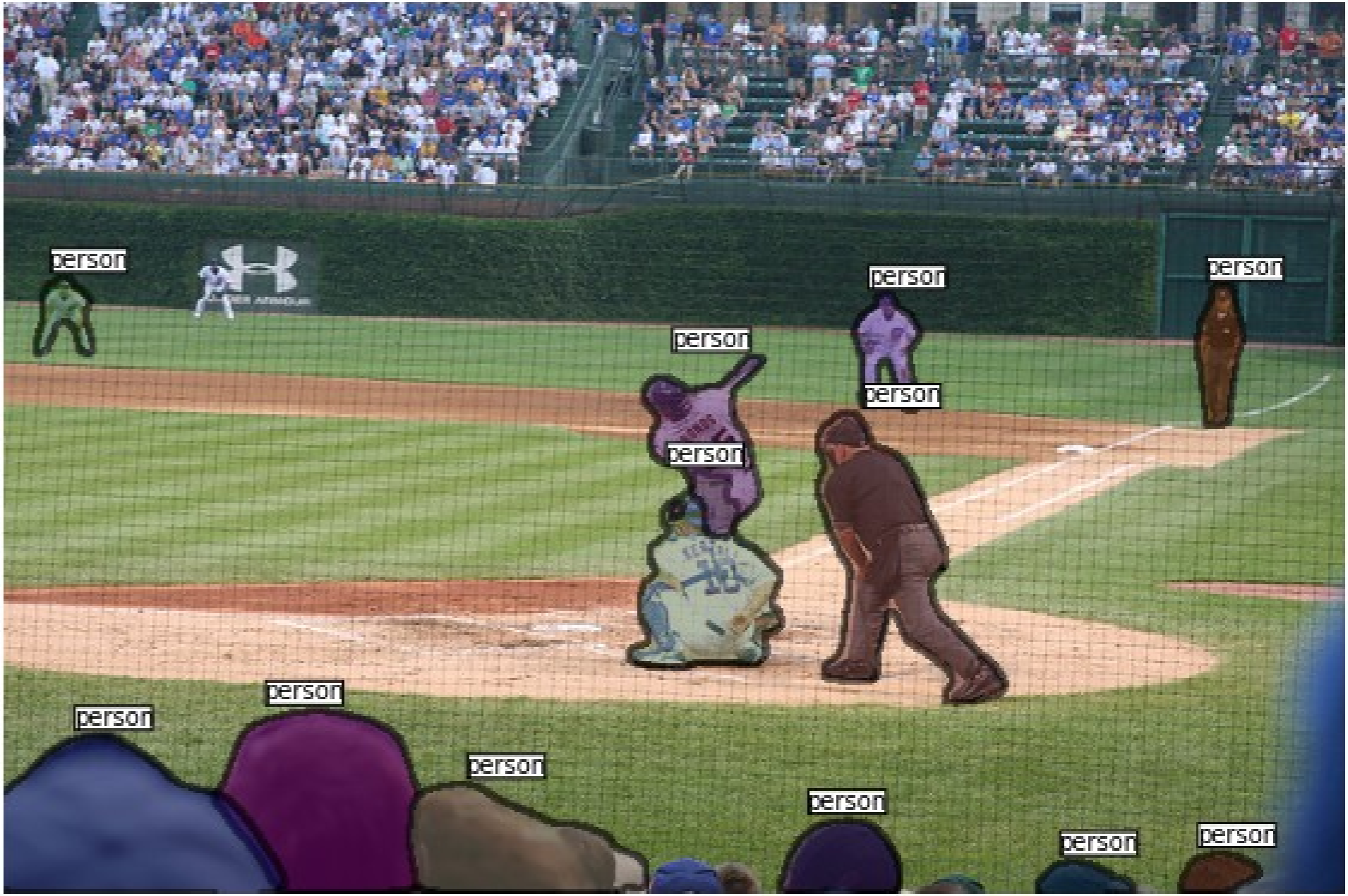
- "Big Sur"

Results



Results

Results


person
person
person
person
person
person
person
person
person
person
person
person
person

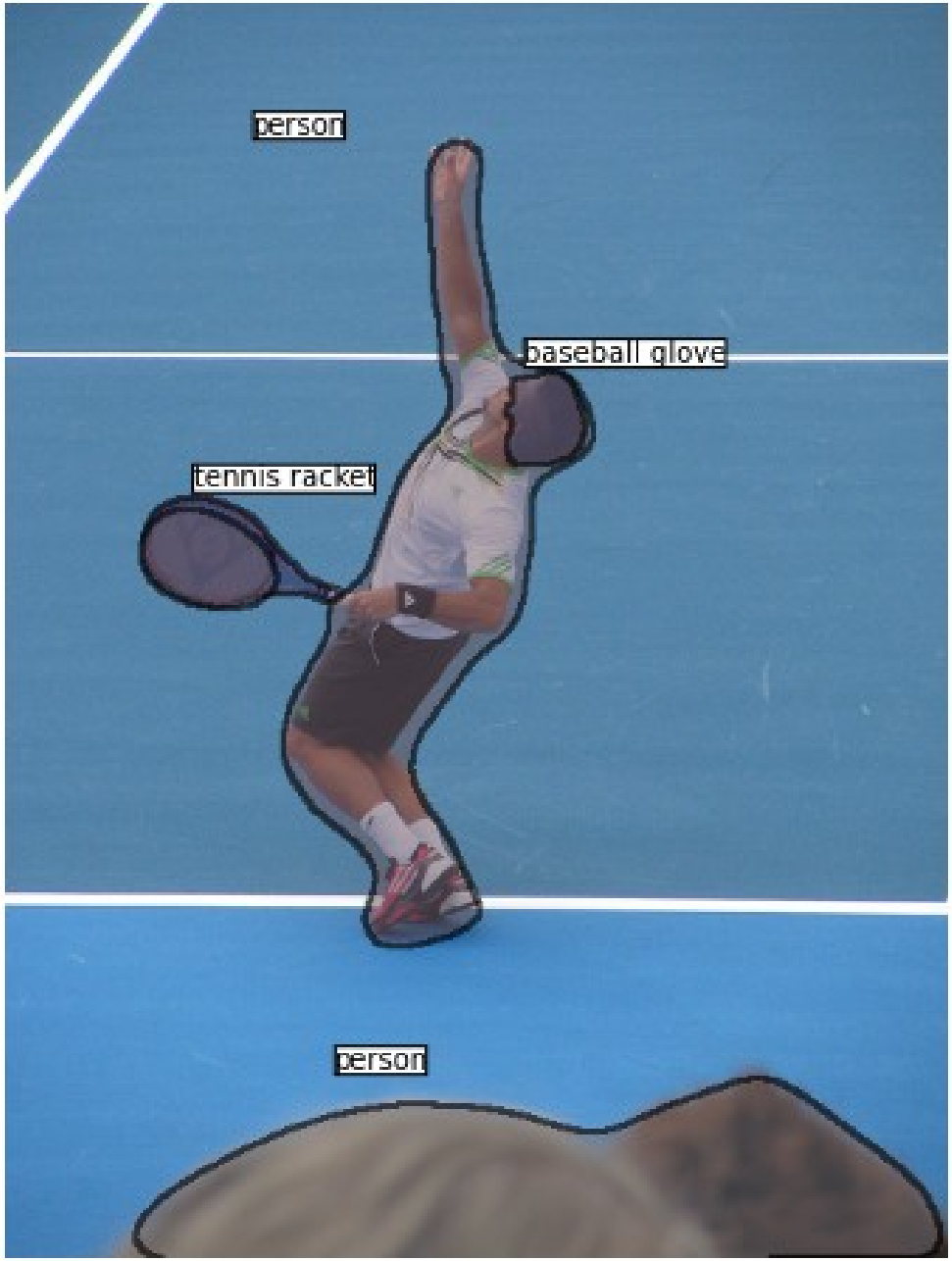
Results

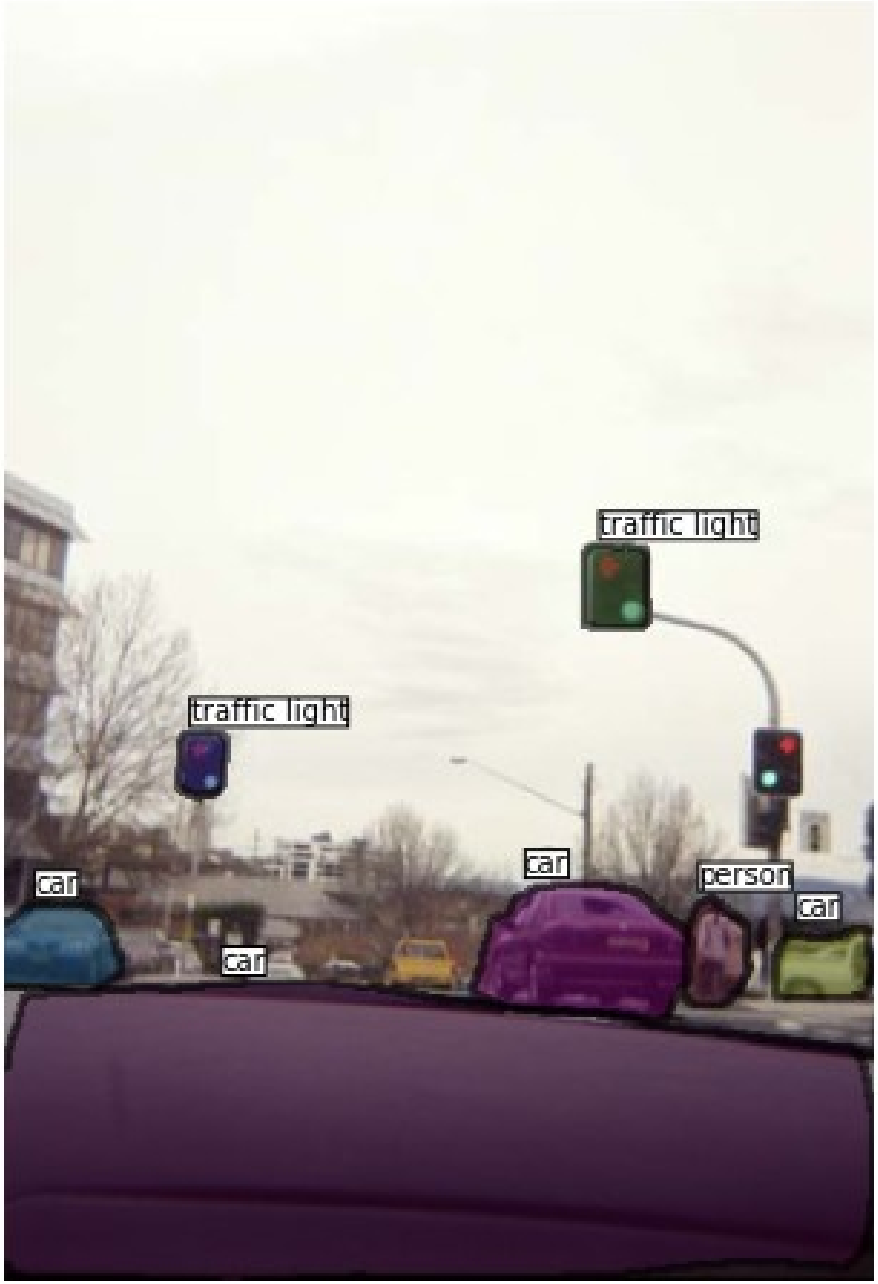
Results

Results

broccoli
broccoli
carrot
broccoli
carrot
broccoli
carrot
broccoli
carrot
broccoli
carrot
broccoli

Mistakes


person
baseball glove
tennis racket
person


traffic light
traffic light
car
car
car
person
car

Results

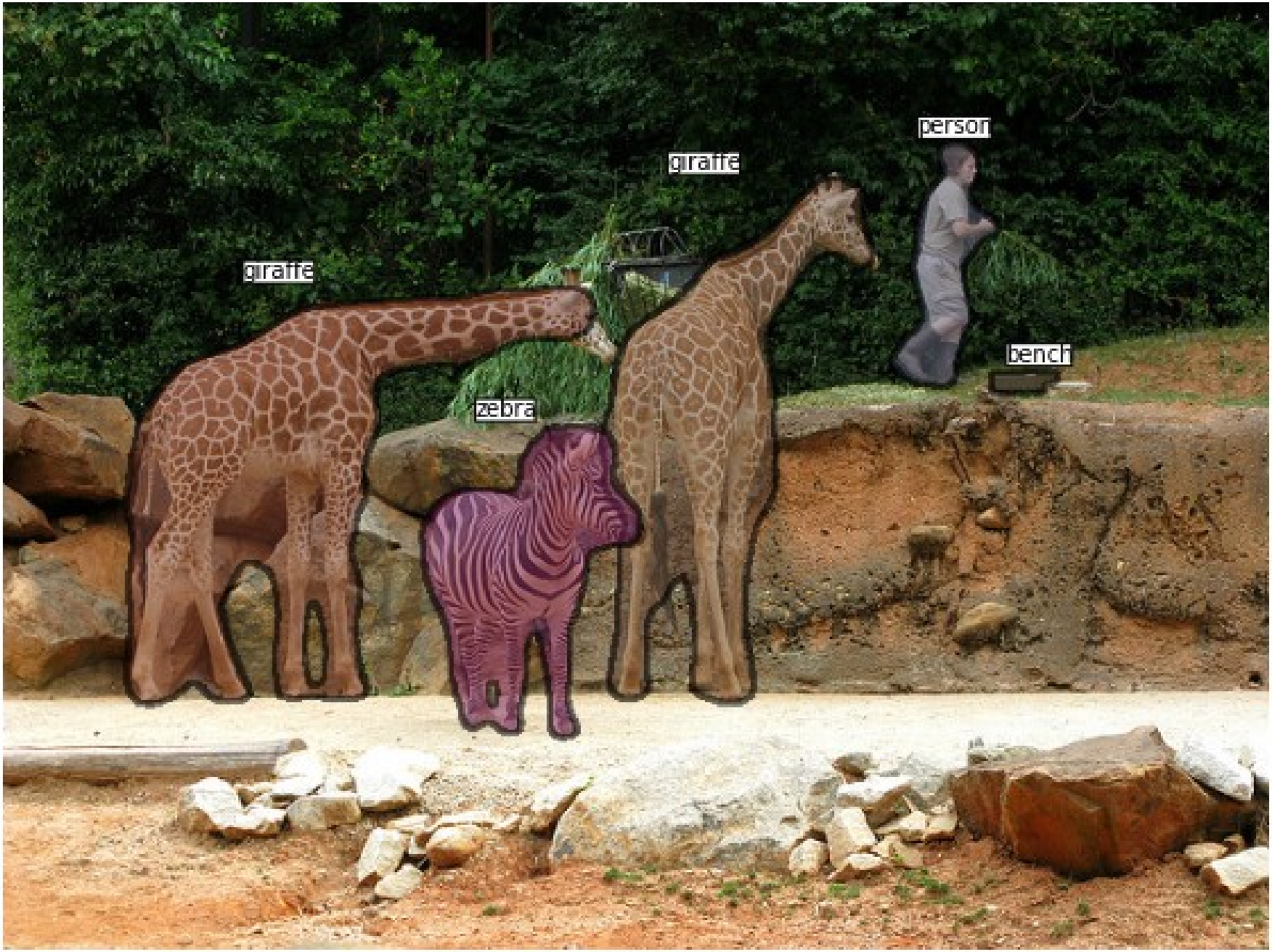
Results

sheep
sheep
sheep
sheep
sheep
sheep
sheep
sheep
sheep
sheep
sheep
sheep
sheep
sheep
sheep

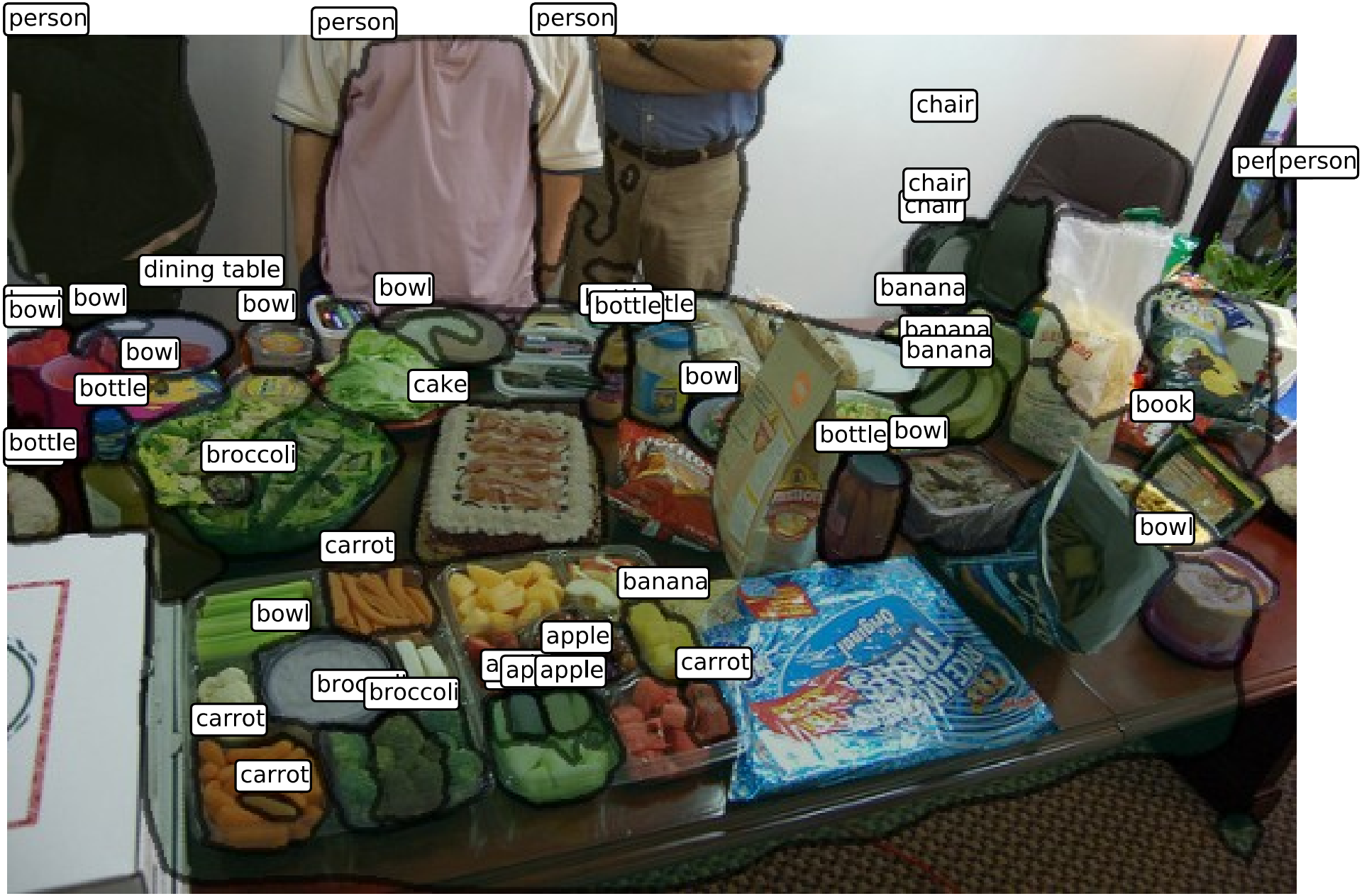
Results

sandwich
donut
donut
donut
donut
donut
donut
donut
donut
donut
donut
donut
donut
donut
donut
donut
donut
donut
donut
donut
donut
donut
donut
donut
donut
donut
donut
donut

Results



Results


person
person
person
chair
chair
chair
person
dining table
bowl
bowl
bowl
bowl
bowl
bottle
bottle
banana
banana
banana
bowl
bottle
cake
bowl
bottle
bowl
book
bottle
broccoli
carrot
bowl
banana
bowl
apple
apple
carrot
broccoli
carrot
carrot

Obstacles to Progress in Artificial Intelligence:

Three missing pieces for AI (besides computation)

Integrating Representation/Deep Learning with Reasoning, Attention, Planning and Memory

- A lot of recent work on reasoning/planning, attention, memory, learning "algorithms".
- Memory-augmented neural nets
- "Differentiable" algorithms

Integrating supervised, unsupervised and reinforcement learning into a single "algorithm".

- Boltzmann Machines would be nice if they worked.
- Stacked What-Where Auto-Encoders, Ladder Networks….

Effective ways to do unsupervised Learning

- Discovering the structure and regularities of the world by observing it and living in it like animals and humans do.

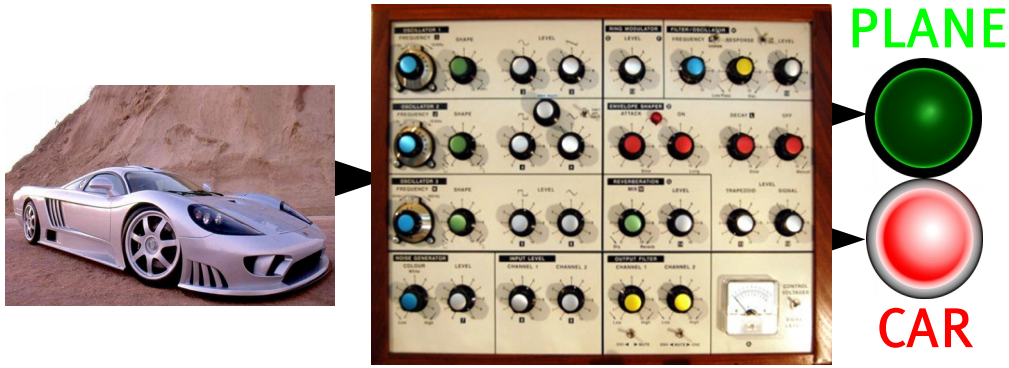
Three Types of Learning

Reinforcement Learning

- The machine predicts a scalar reward given once in a while.
- A few bits for some samples

Supervised Learning

- The machine predicts a category or a few numbers for each input
- 10→10,000 bits per sample



Unsupervised Learning

- The machine predicts any part of its input for any observed part.
- Predicts future frames in videos
- Millions of bits per sample

How Much Information Does the Machine Need to Predict?

Y LeCun

Reinforcement Learning (cherry)

- The machine predicts a scalar reward given once in a while.
- **A few bits for some samples**

Supervised Learning (icing)

- The machine predicts a category or a few numbers for each input
- **10→10,000 bits per sample**

Unsupervised Learning (cake)

- The machine predicts any part of its input for any observed part.
- Predicts future frames in videos
- **Millions of bits per sample**

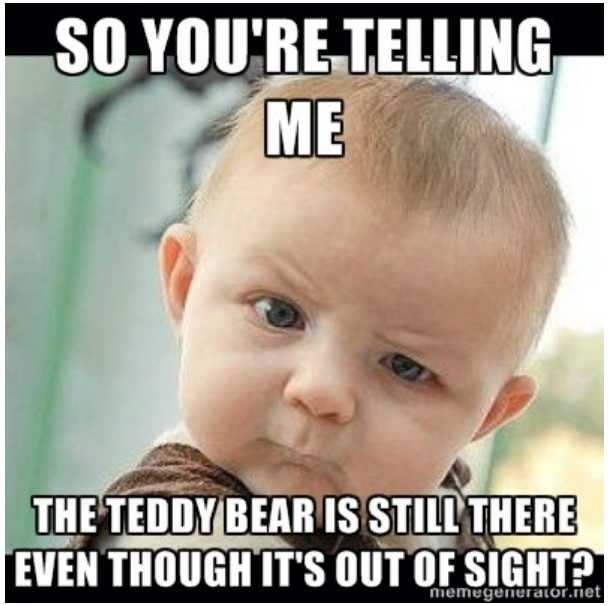
Unsupervised Learning is the "Dark Matter" of AI

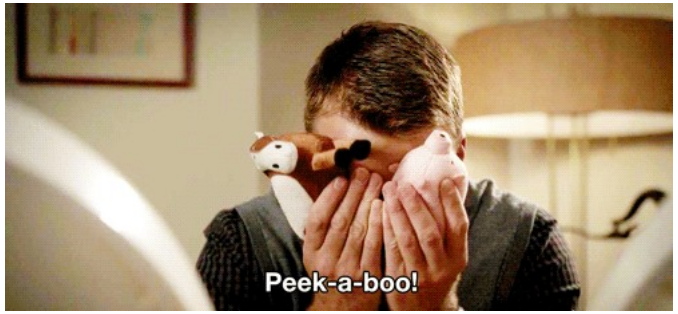
Y LeCun

Most of the learning performed by animals and humans is unsupervised

We learn how the world works by observing it

- We learn that the world is 3-dimensional
- We learn that objects can move independently of each other
- We learn that object permanence
- We learn to predict what the world will look like one second or one hour from now.





We build a model of the world through predictive unsupervised learning

This predictive model gives us "common sense"

Unsupervised learning discovers regularities in the world.

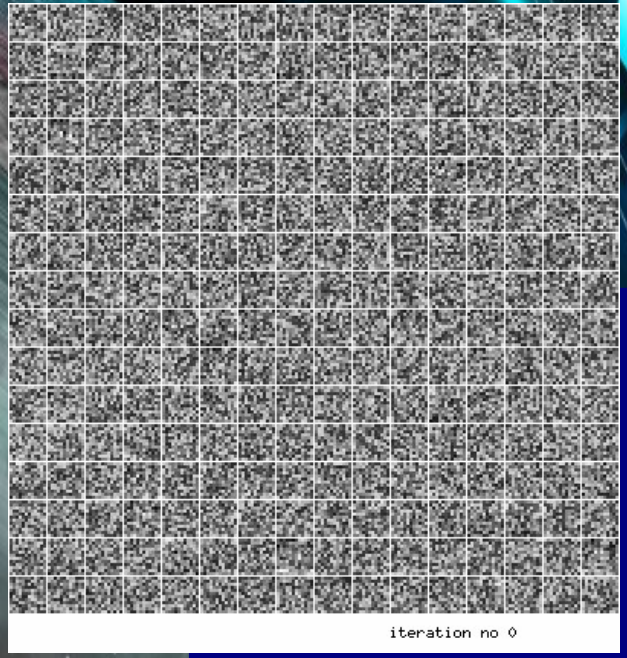
Common Sense through Unsupervised Learning

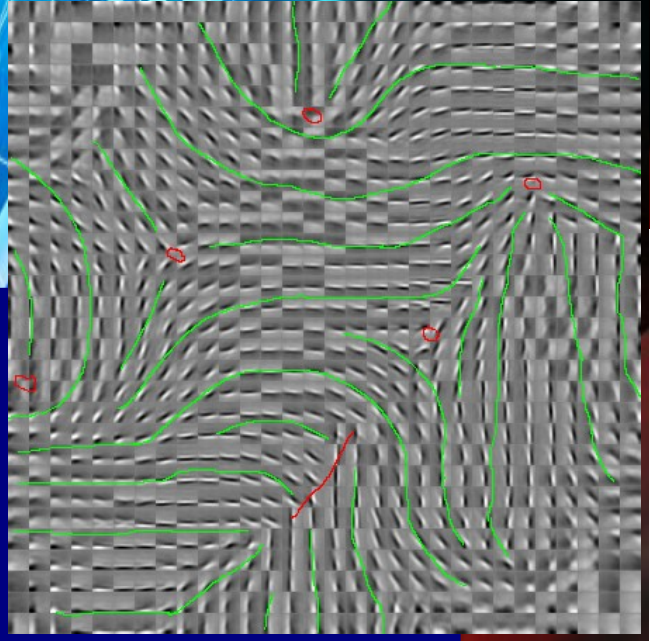
Learning a predictive model of the world gives us common sense.

If I say: "Bernhard picks up his bag and leaves the room"

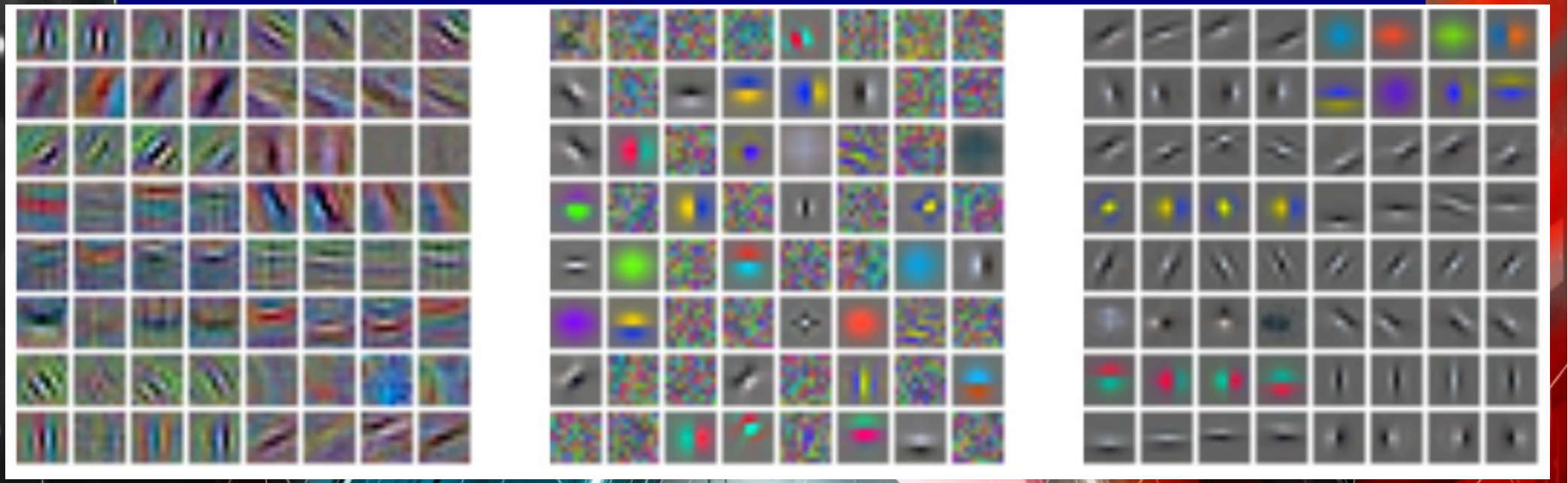
You can infer:

- Bernhard stood up, extended his arm, walked towards the door, opened the door, walked out.
- He and his bag are not in the room anymore.
- He probably didn't dematerialize of fly out.





Unsupervised Learning



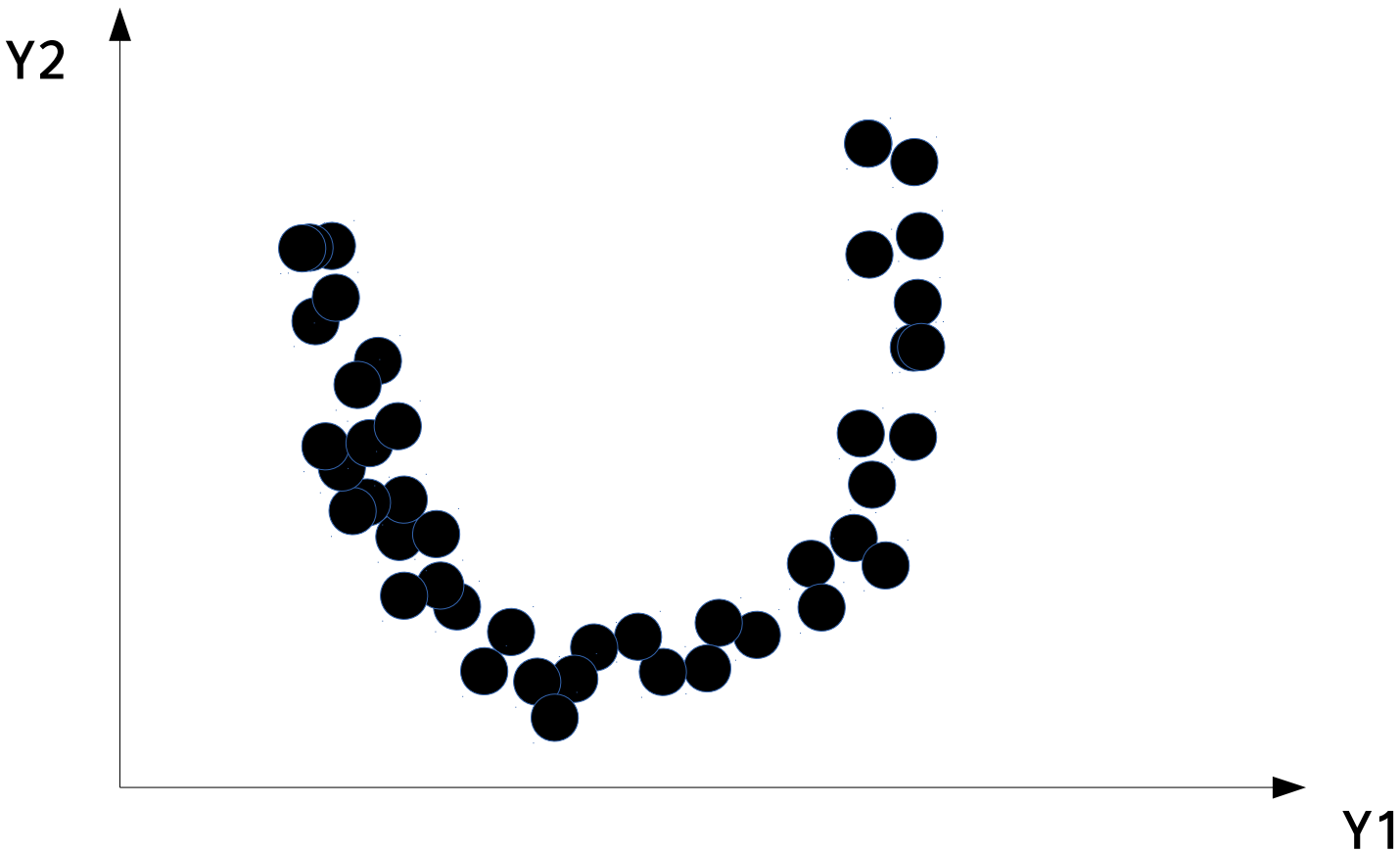
How do we do unspervised learning?

- Unsupervised learning can be based on reconstruction or prediction
  - Reconstruction is just prediction with delay 0
  - It seems qualitatively different from prediction, but it's not
  - Because a good way to predict the future is to copy the present
- How do we deal with the fact that the world is only partially predictable?
  - Model the future as a **distribution**: **hahaha, good luck with that!**
  - Modeling a distribution works when the world is discrete (e.g. words)
  - It doesn't work when the world is continuous or compositional.
- Even if the world were noiseless and quasi deterministic, our current methods would fail
  - Imagine the world follows a trajectory on a low-dimensional manifold with a few discrete choices at certain places.
  - Our current methods would still fail.
  - Our failure to do unsupervised learning has nothing to do with our inability to model **probability distributions** in high-dim spaces.
  - **It has to do with our inability to capture complex regularities.**
  - **I'm allergic to sampling anyway.**

Energy-Based Unsupervised Learning

Energy-Based Unsupervised Learning

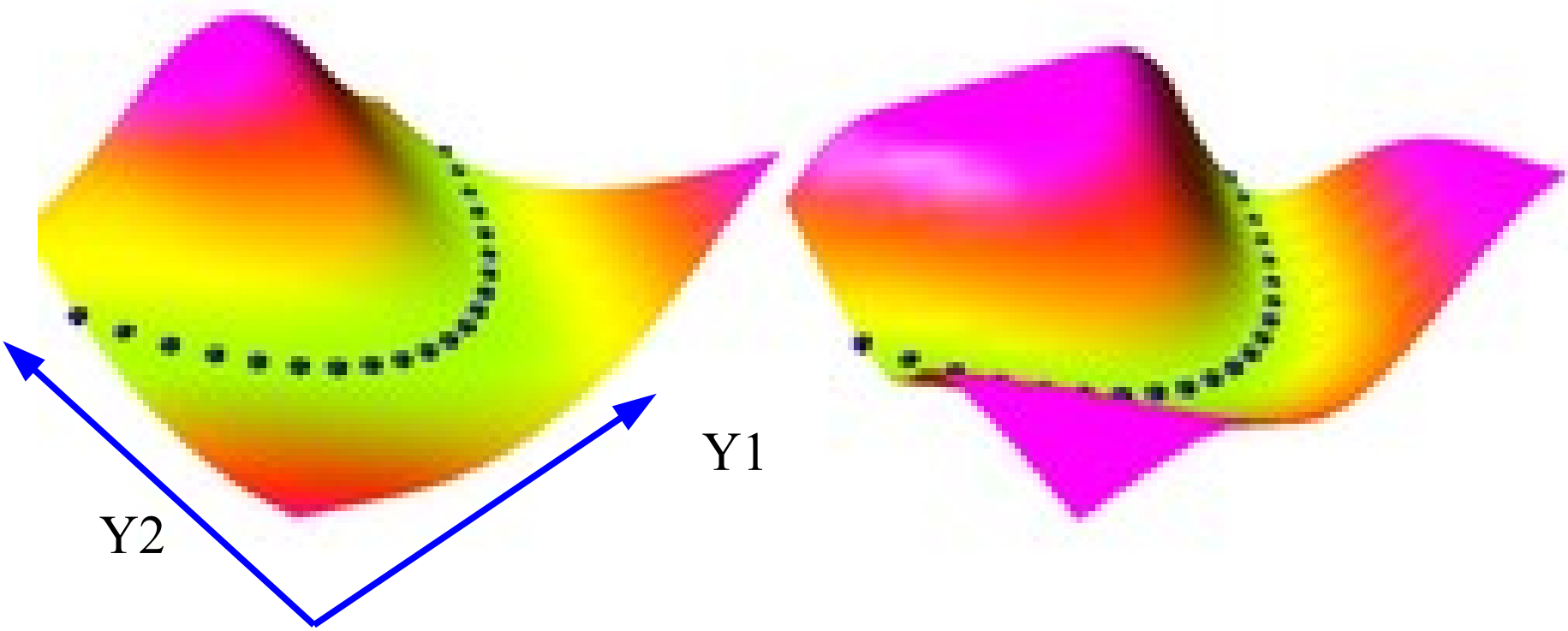
- Learning an energy function (or contrast function) that takes
  - Low values on the data manifold
  - Higher values everywhere else



Capturing Dependencies Between Variables with an Energy Function

- The energy surface is a "contrast function" that takes low values on the data manifold, and higher values everywhere else
  - Special case: energy = negative log density
  - Example: the samples live in the manifold

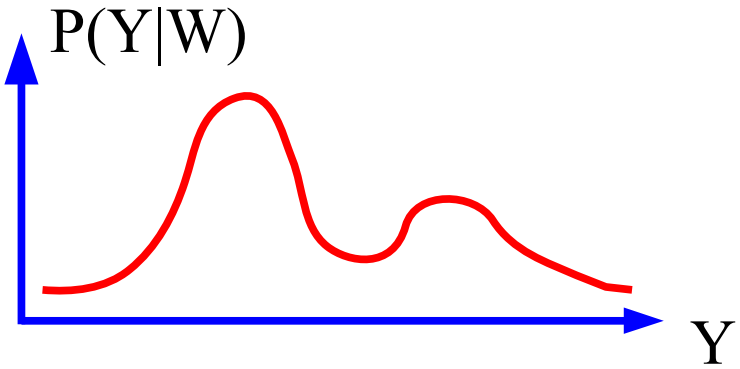
$$Y_2 = (Y_1)^2$$



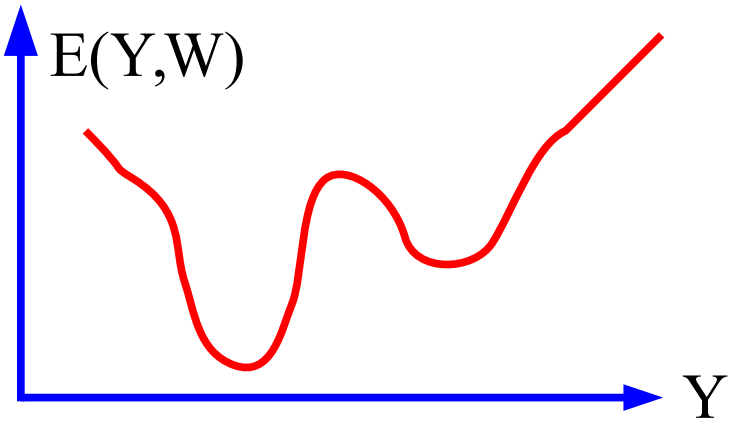
Transforming Energies into Probabilities (if necessary)

- The energy can be interpreted as an unnormalized negative log density
- Gibbs distribution: Probability proportional to exp(-energy)
  - Beta parameter is akin to an inverse temperature
- Don't compute probabilities unless you absolutely have to
  - Because the denominator is often intractable

$$P(Y|W) = \frac{e^{-\beta E(Y,W)}}{\int_y e^{-\beta E(y,W)}}$$

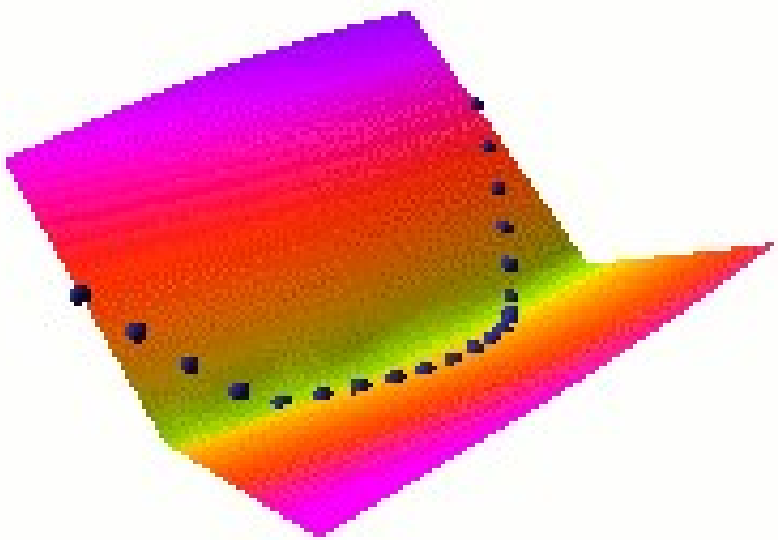


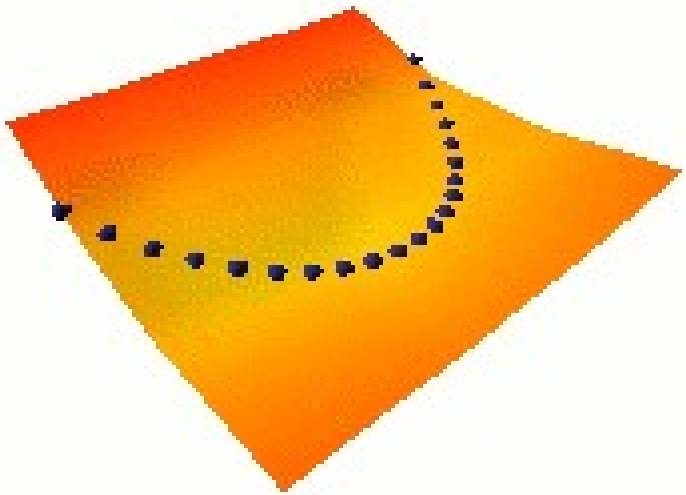
$$E(Y,W) \propto -\log P(Y|W)$$



Learning the Energy Function

- parameterized energy function E(Y,W)
  - Make the energy low on the samples
  - Make the energy higher everywhere else
  - Making the energy low on the samples is easy
  - But how do we make it higher everywhere else?





Eight Strategies to Shape the Energy Function

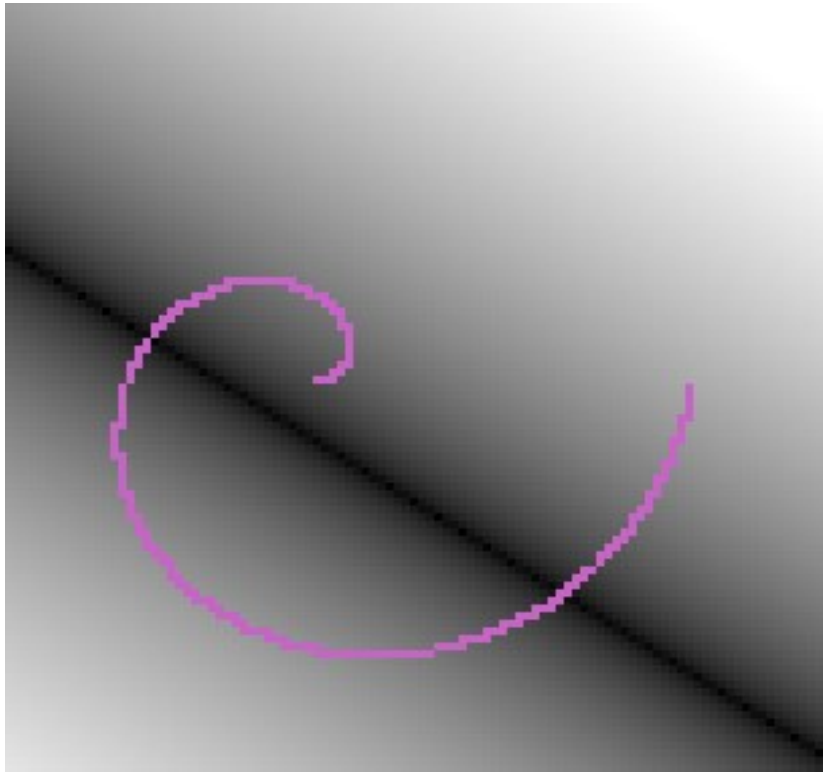
- 1. build the machine so that the volume of low energy stuff is constant
  - PCA, K-means, GMM, square ICA
- 2. push down of the energy of data points, push up everywhere else
  - Max likelihood (needs tractable partition function)
- 3. push down of the energy of data points, push up on chosen locations
  - contrastive divergence, Ratio Matching, Noise Contrastive Estimation, Minimum Probability Flow
- 4. minimize the gradient and maximize the curvature around data points
  - score matching
- 5. train a dynamical system so that the dynamics goes to the manifold
  - denoising auto-encoder
- 6. use a regularizer that limits the volume of space that has low energy
  - Sparse coding, sparse auto-encoder, PSD
- 7. if $E(Y) = \|Y - G(Y)\|^2$, make $G(Y)$ as "constant" as possible.
  - Contracting auto-encoder, saturating auto-encoder
- 8. Adversarial training: generator tries to fool real/synthetic classifier.

#1: constant volume of low energy

- 1. build the machine so that the volume of low energy stuff is constant
  - PCA, K-means, GMM, square ICA...

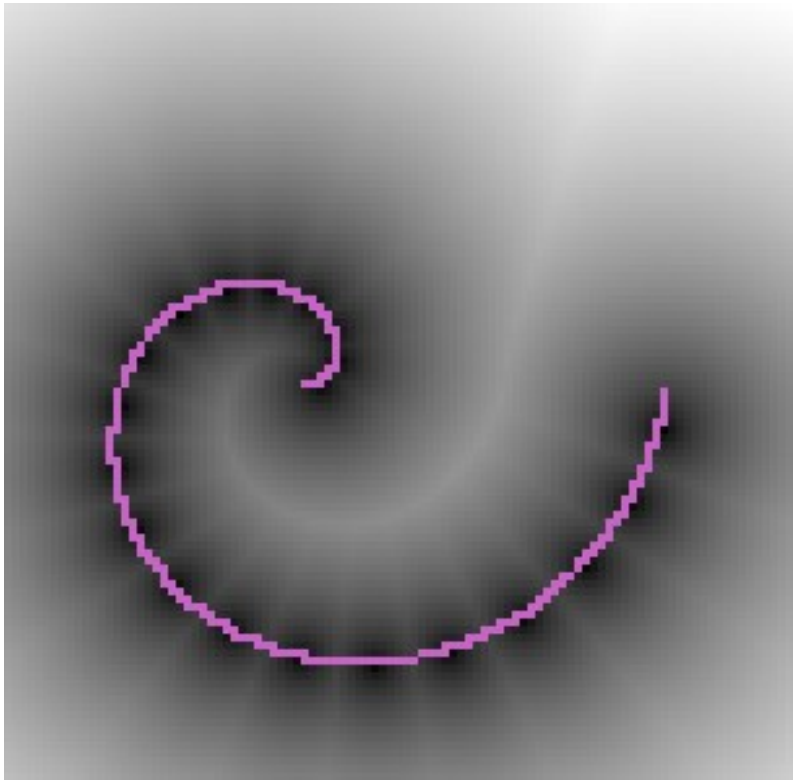
PCA

$$E(Y)=\|W^T WY - Y\|^2$$



K-Means,
Z constrained to 1-of-K code

$$E(Y)=min_z \sum_i \|Y - W_i Z_i\|^2$$



#2: push down of the energy of data points, push up everywhere else

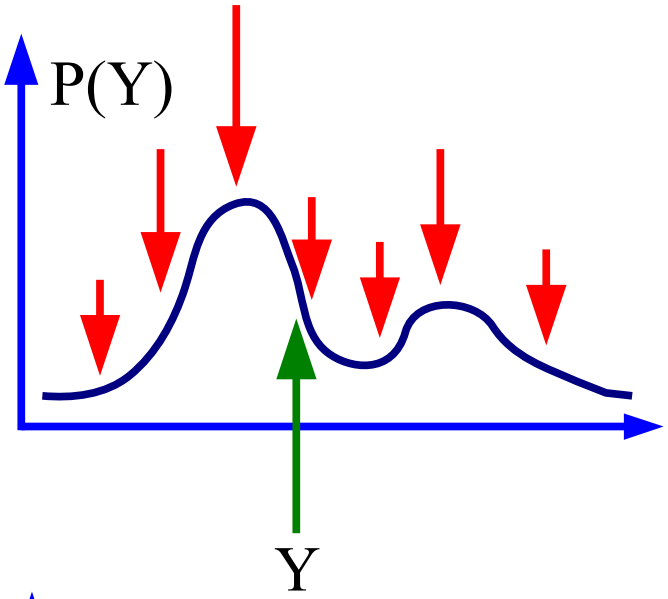
Max likelihood (requires a tractable partition function)

Maximizing P(Y|W) on training samples

make this big

$$P(Y|W) = \frac{e^{-\beta E(Y,W)}}{\int_y e^{-\beta E(y,W)}}$$

make this small

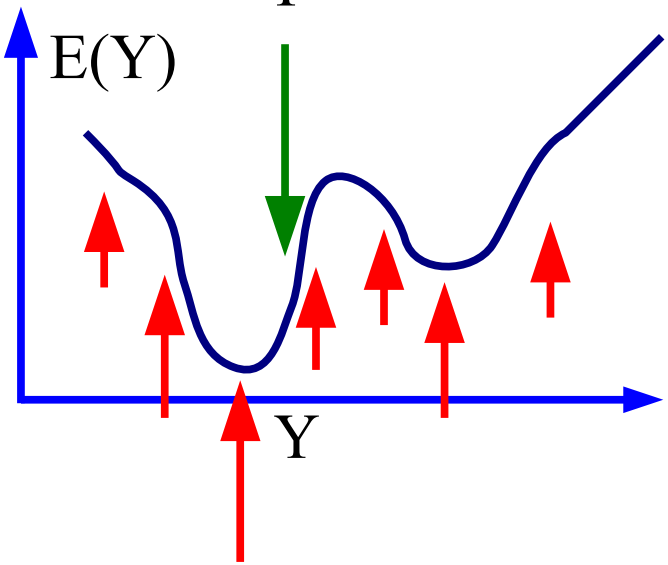


Minimizing -log P(Y,W) on training samples

$$L(Y,W) = E(Y,W) + \frac{1}{\beta}\log\int_y e^{-\beta E(y,W)}$$

make this small

make this big



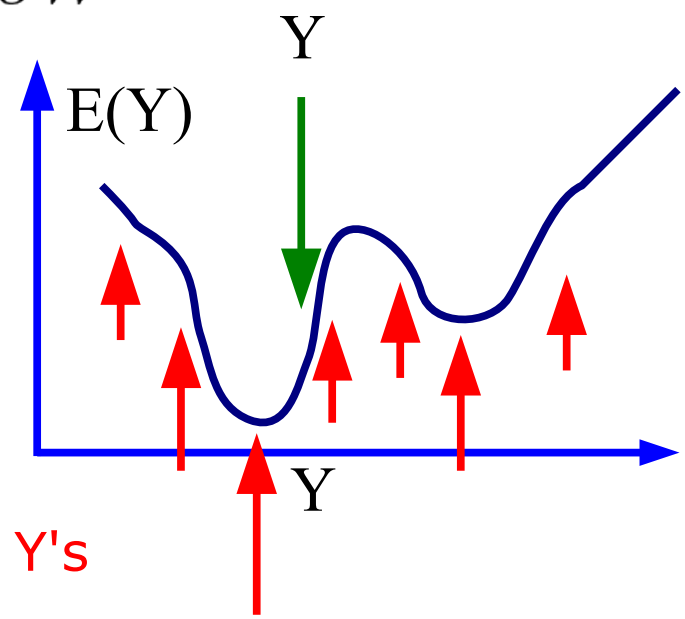
#2: push down of the energy of data points, push up everywhere else

Gradient of the negative log-likelihood loss for one sample Y:

$$\frac{\partial L(Y,W)}{\partial W} = \frac{\partial E(Y,W)}{\partial W} - \int_y P(y|W)\frac{\partial E(y,W)}{\partial W}$$

Gradient descent:

$$W \leftarrow W - \eta \frac{\partial L(Y,W)}{\partial W}$$



Pushes down on the energy of the samples

Pulls up on the energy of low-energy Y's

$$W \leftarrow W - \eta \frac{\partial E(Y,W)}{\partial W} + \eta \int_y P(y|W)\frac{\partial E(y,W)}{\partial W}$$

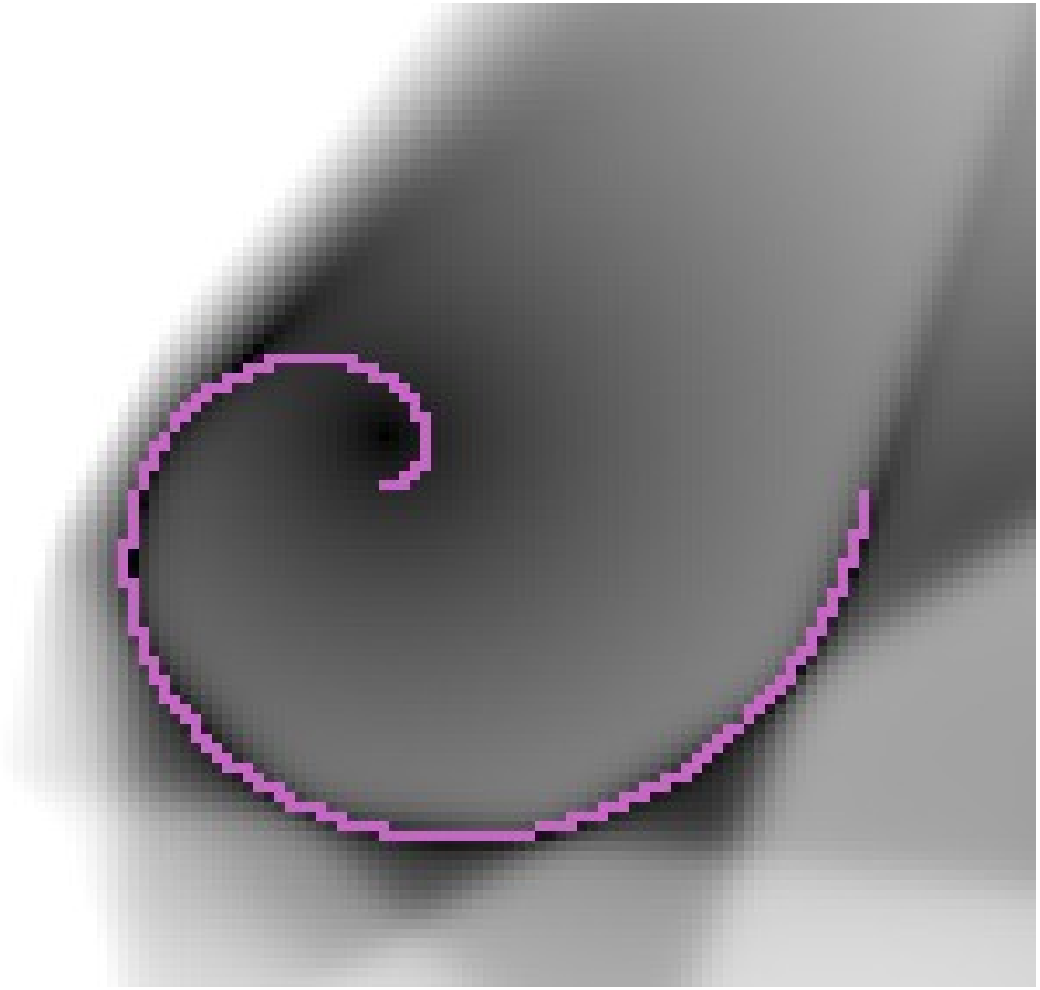
#3. push down of the energy of data points, push up on chosen locations

- contrastive divergence, Ratio Matching, Noise Contrastive Estimation, Minimum Probability Flow
- Contrastive divergence: basic idea
  - Pick a training sample, lower the energy at that point
  - From the sample, move down in the energy surface with noise
  - Stop after a while
  - Push up on the energy of the point where we stopped
  - This creates grooves in the energy surface around data manifolds
  - CD can be applied to any energy function (not just RBMs)
- Persistent CD: use a bunch of "particles" and remember their positions
  - Make them roll down the energy surface with noise
  - Push up on the energy wherever they are
  - Faster than CD
- RBM

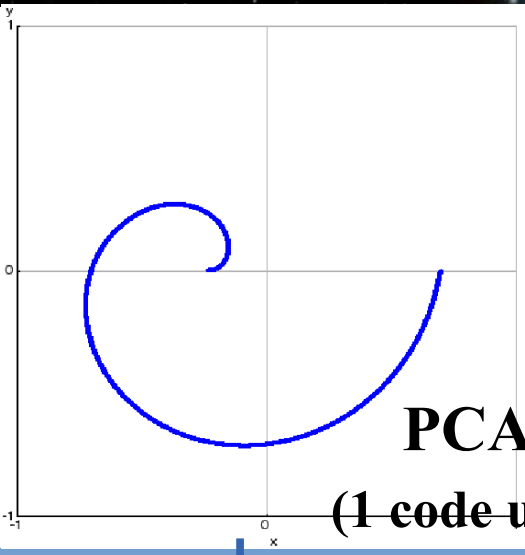
$$E(Y,Z) = -Z^T W Y \qquad E(Y) = -\log \sum_z e^{Z^T W Y}$$

#6. use a regularizer that limits the volume of space that has low energy

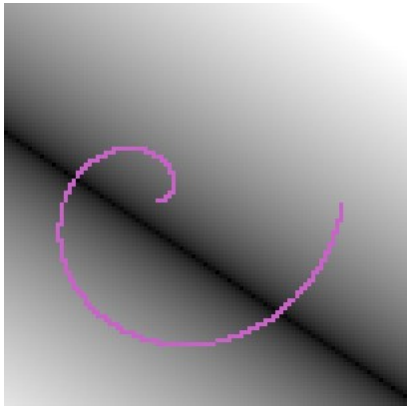
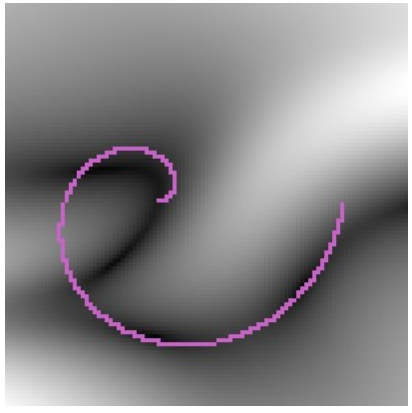
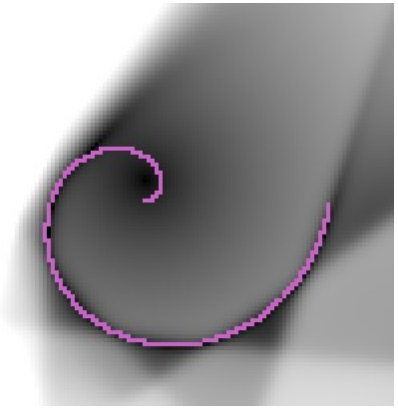
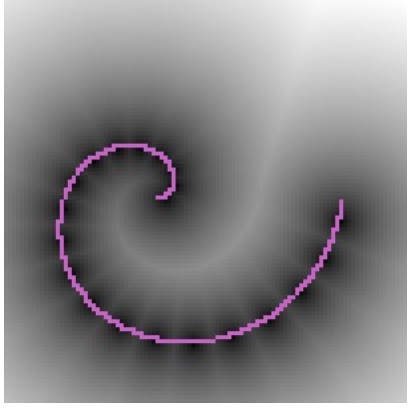
- Sparse coding, sparse auto-encoder, Predictive Saprse Decomposition



Energy Functions of Various Methods



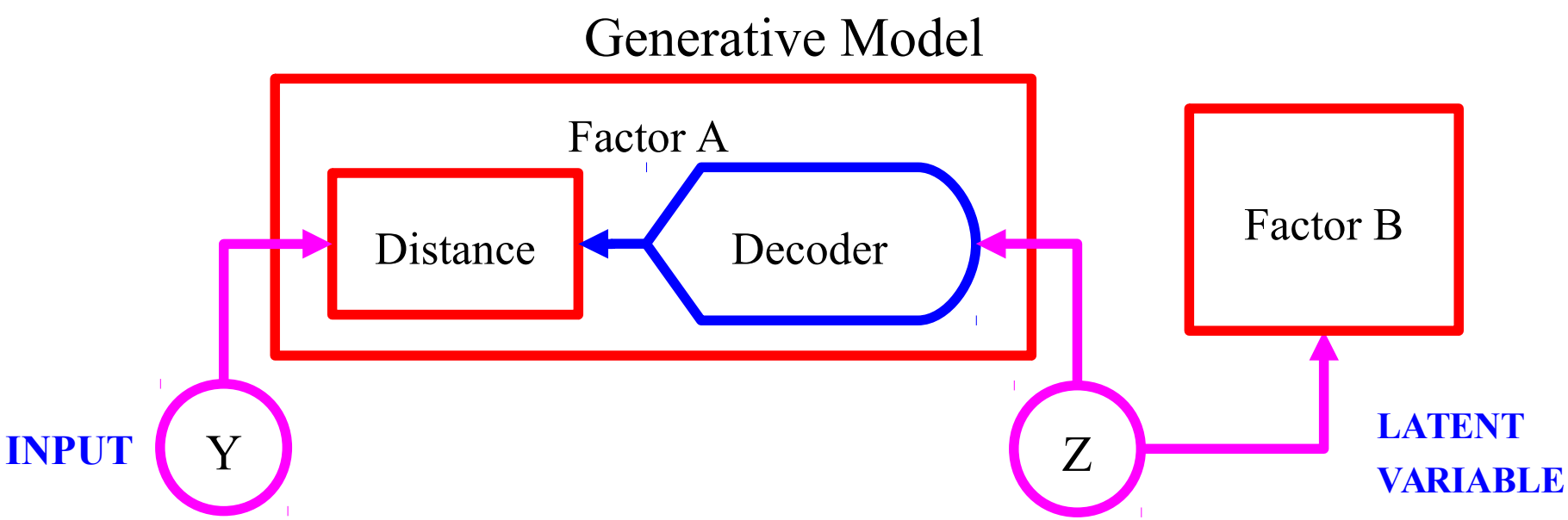
- 2 dimensional toy dataset: spiral
- Visualizing energy surface
  - (black = low, white = high)

	PCA (1 code unit)	autoencoder (1 code unit)	sparse coding (20 code units)	K-Means (20 code units)
encoder	$W'Y$	$\sigma(W_e Y)$	$\sigma(W_e Z)$	–
decoder	WZ	$W_d Z$	$W_d Z$	WZ
energy	$\|\|Y-WZ\|\|^2$	$\|\|Y-WZ\|\|^2$	$\|\|Y-WZ\|\|^2$	$\|\|Y-WZ\|\|^2$
loss	$F(Y)$	$F(Y)$	$F(Y)$	$F(Y)$
pull-up	dimens.	dimens.	sparsity	1-of-N code
				

Sparse Modeling,
Sparse Auto-Encoders,
Predictive Sparse Decomposition
LISTA

How to Speed Up Inference in a Generative Model?

- Factor Graph with an asymmetric factor
- Inference Z → Y is easy
  - Run Z through deterministic decoder, and sample Y
- Inference Y → Z is hard, particularly if Decoder function is many-to-one
  - MAP: minimize sum of two factors with respect to Z
  - Z* = argmin_z Distance[Decoder(Z), Y] + FactorB(Z)
- Examples: K-Means (1of K), Sparse Coding (sparse), Factor Analysis

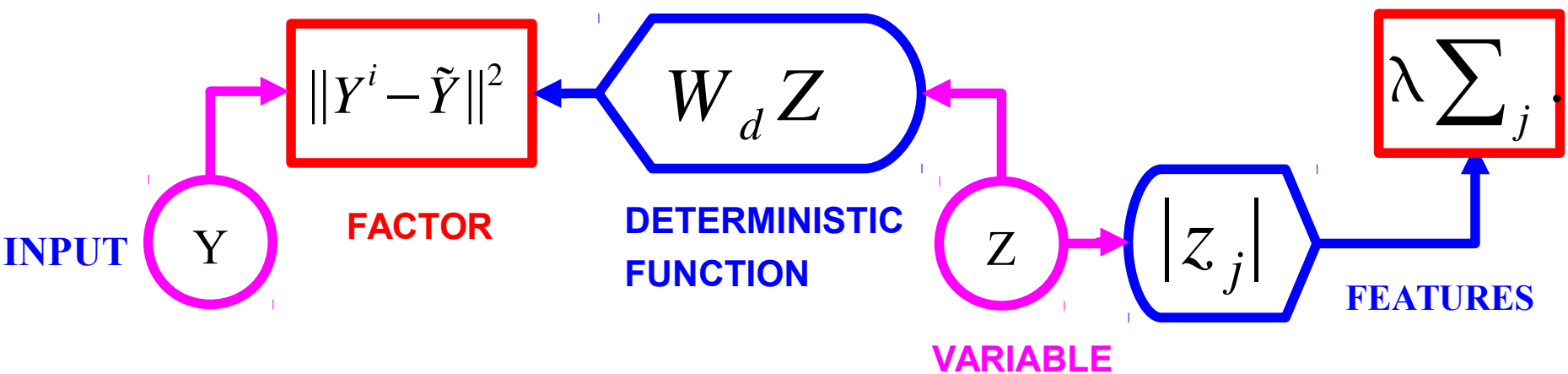


Sparse Coding & Sparse Modeling

[Olshausen & Field 1997]

- Sparse linear reconstruction
- Energy = reconstruction_error + code_prediction_error + code_sparsity

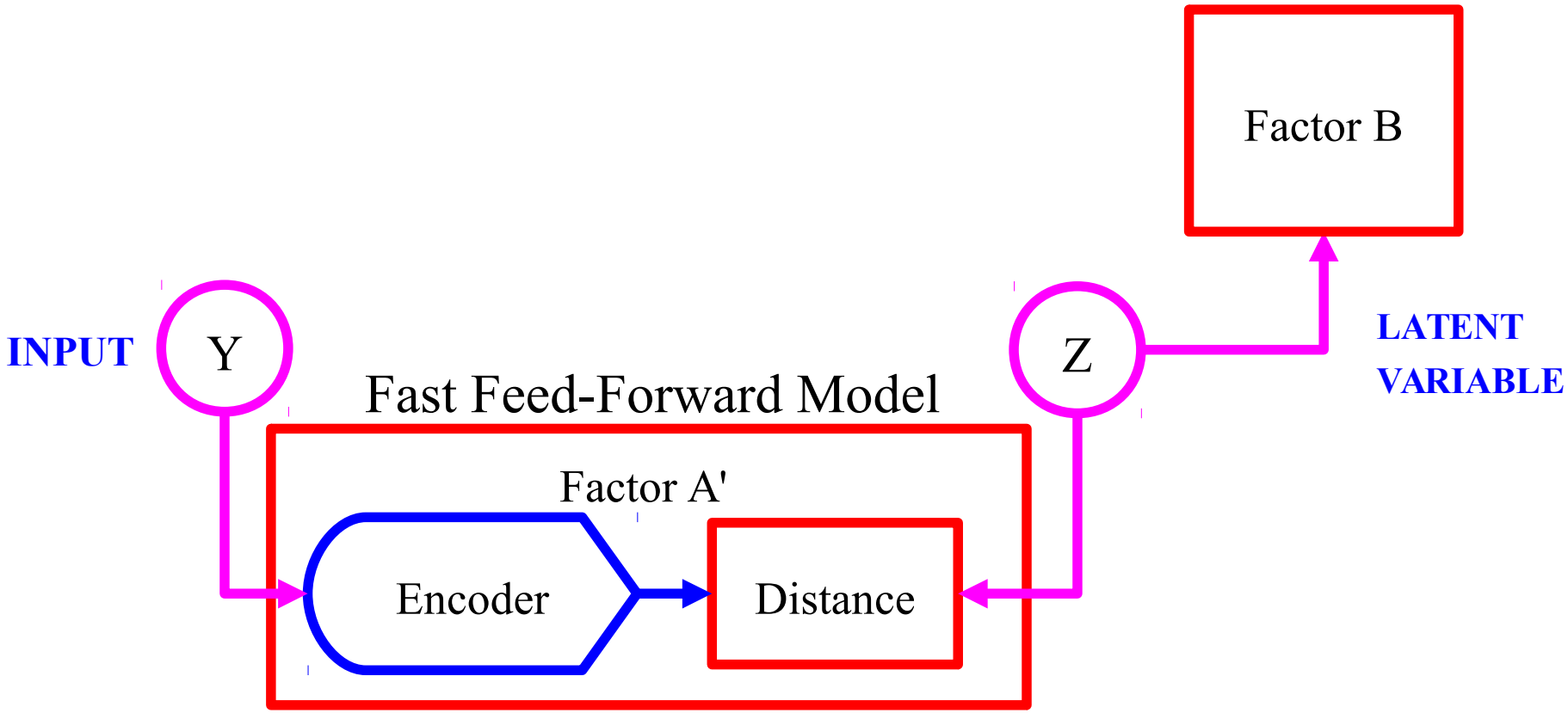
$$E(Y^i, Z) = \|Y^i - W_d Z\|^2 + \lambda \sum_j |z_j|$$



- Inference is slow $Y \rightarrow \hat{Z} = argmin_Z E(Y, Z)$

Encoder Architecture

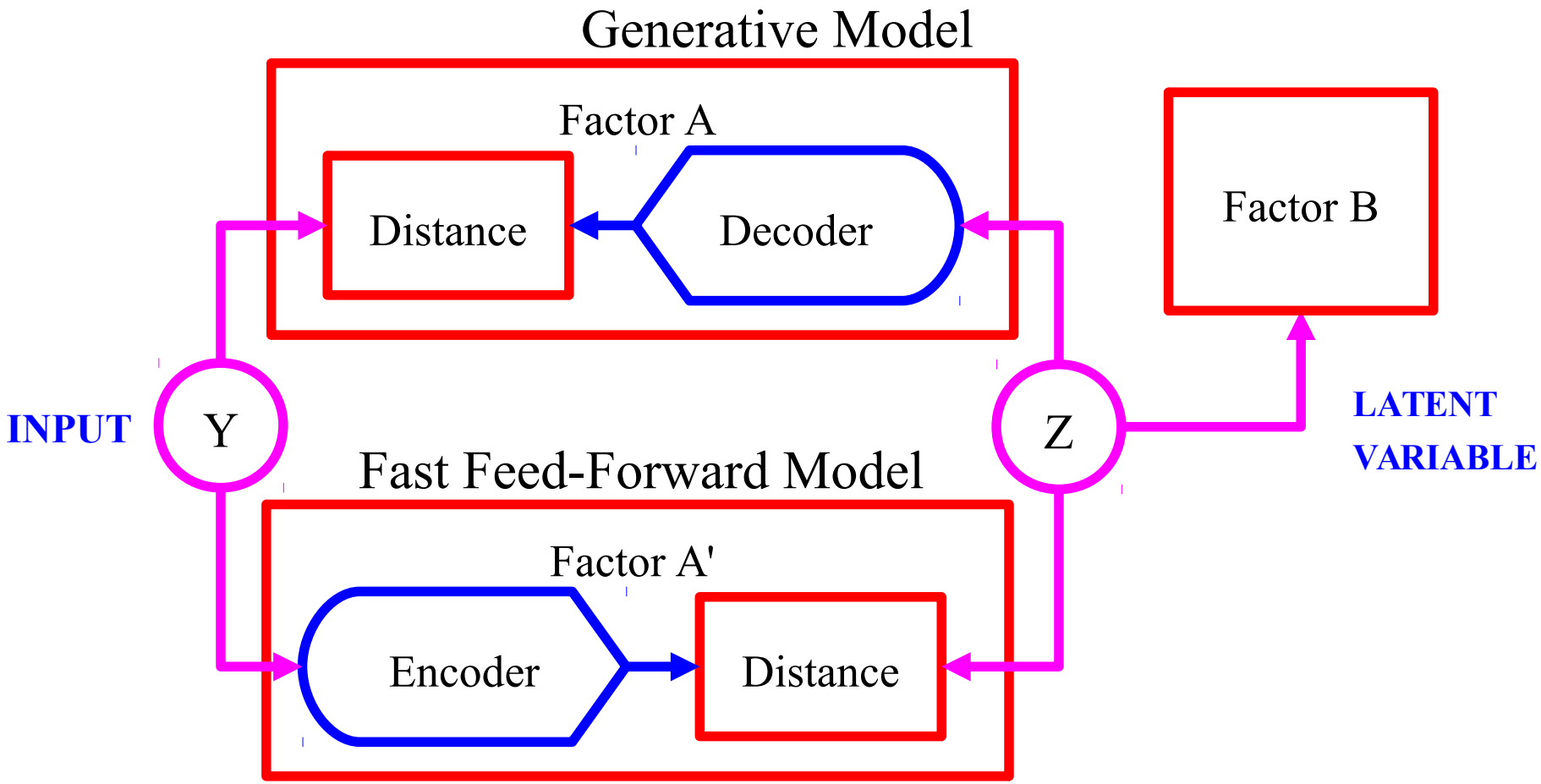
Examples: most ICA models, Product of Experts



Encoder-Decoder Architecture

[Kavukcuoglu, Ranzato, LeCun, rejected by every conference, 2008-2009]

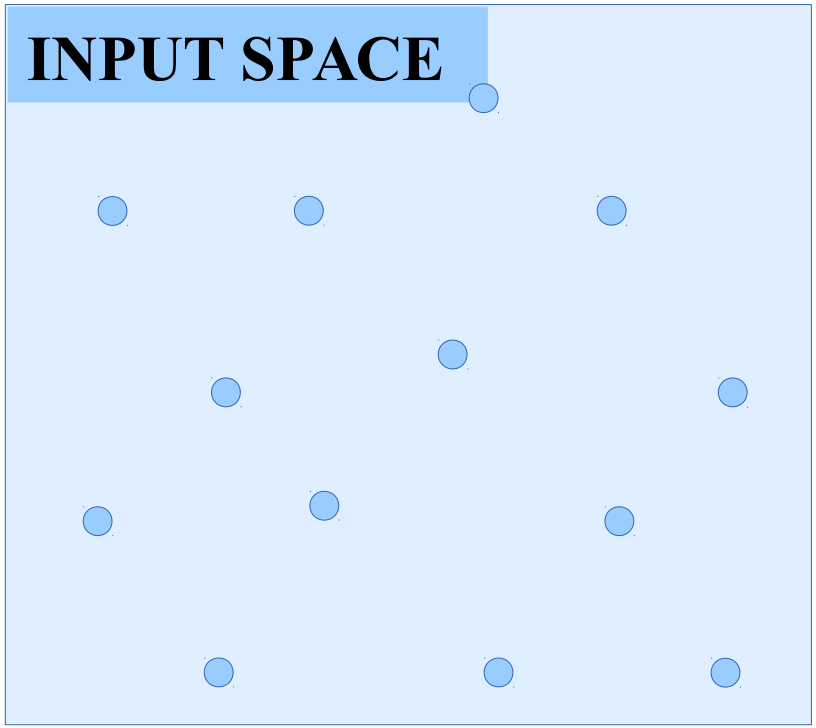
- Train a "simple" feed-forward function to predict the result of a complex optimization on the data points of interest

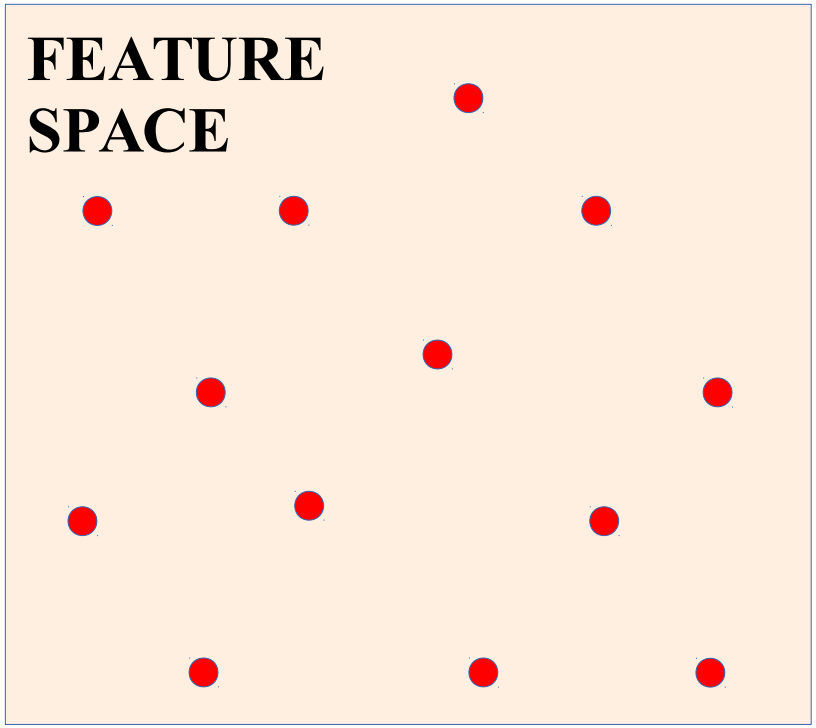


- 1. Find optimal Zi for all Yi; 2. Train Encoder to predict Zi from Yi

Why Limit the Information Content of the Code?

- Training sample
- Input vector which is NOT a training sample
- Feature vector

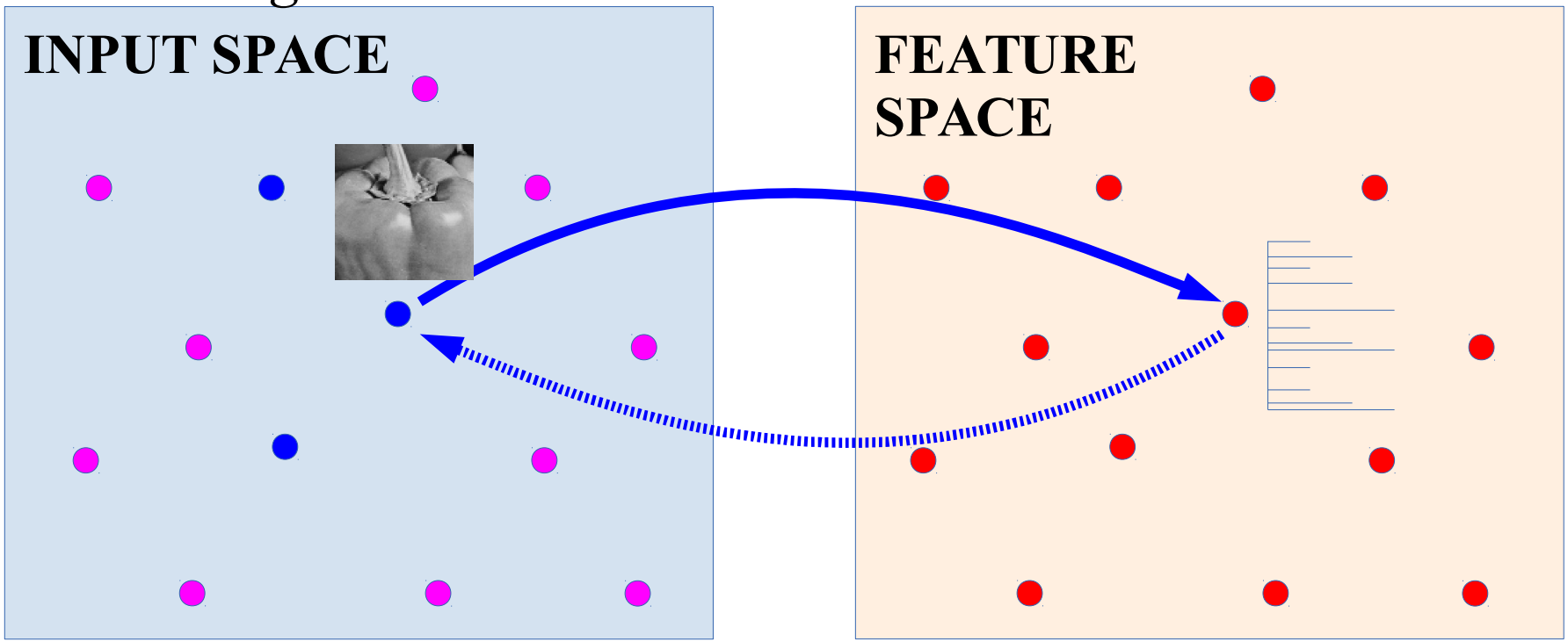




Why Limit the Information Content of the Code?

- **Training sample**
- **Input vector which is NOT a training sample**
- **Feature vector**

Training based on minimizing the reconstruction error over the training set

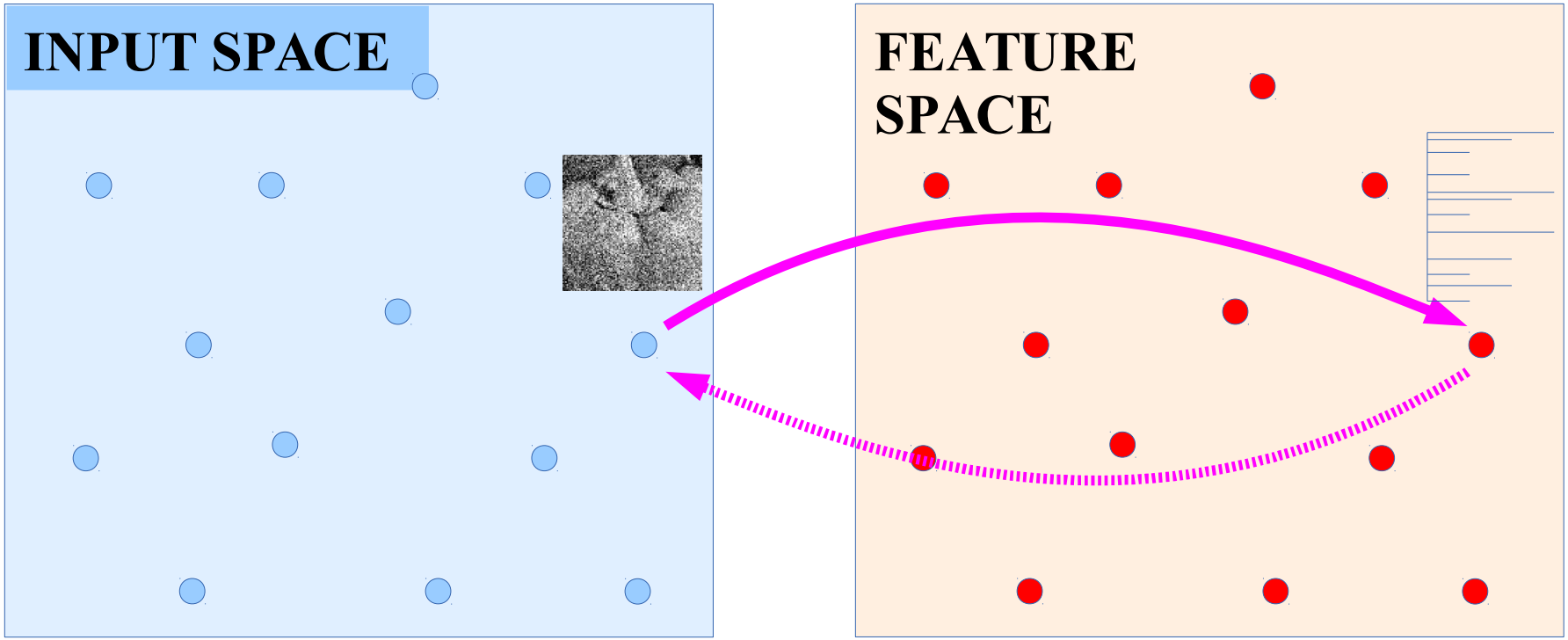


Why Limit the Information Content of the Code?

- **Training sample**
- **Input vector which is NOT a training sample**
- **Feature vector**

BAD: machine does not learn structure from training data!!

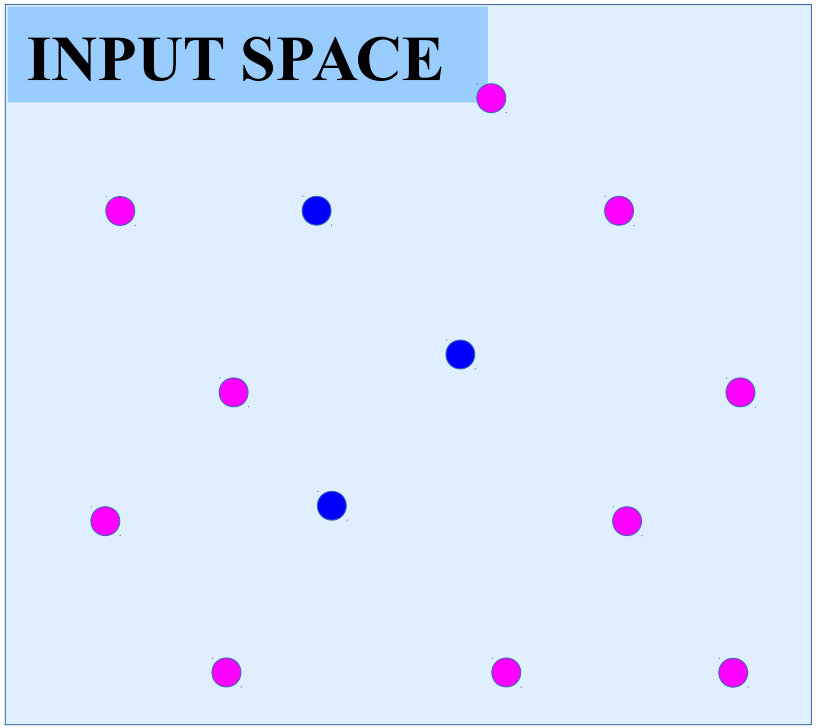
It just copies the data.

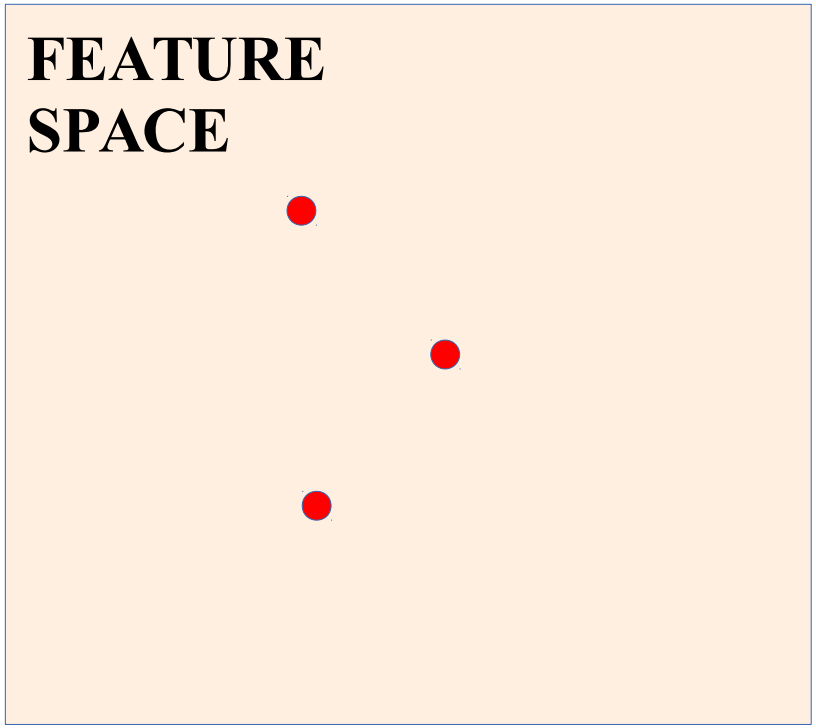


Why Limit the Information Content of the Code?

- Training sample
- Input vector which is NOT a training sample
- Feature vector

IDEA: reduce number of available codes.

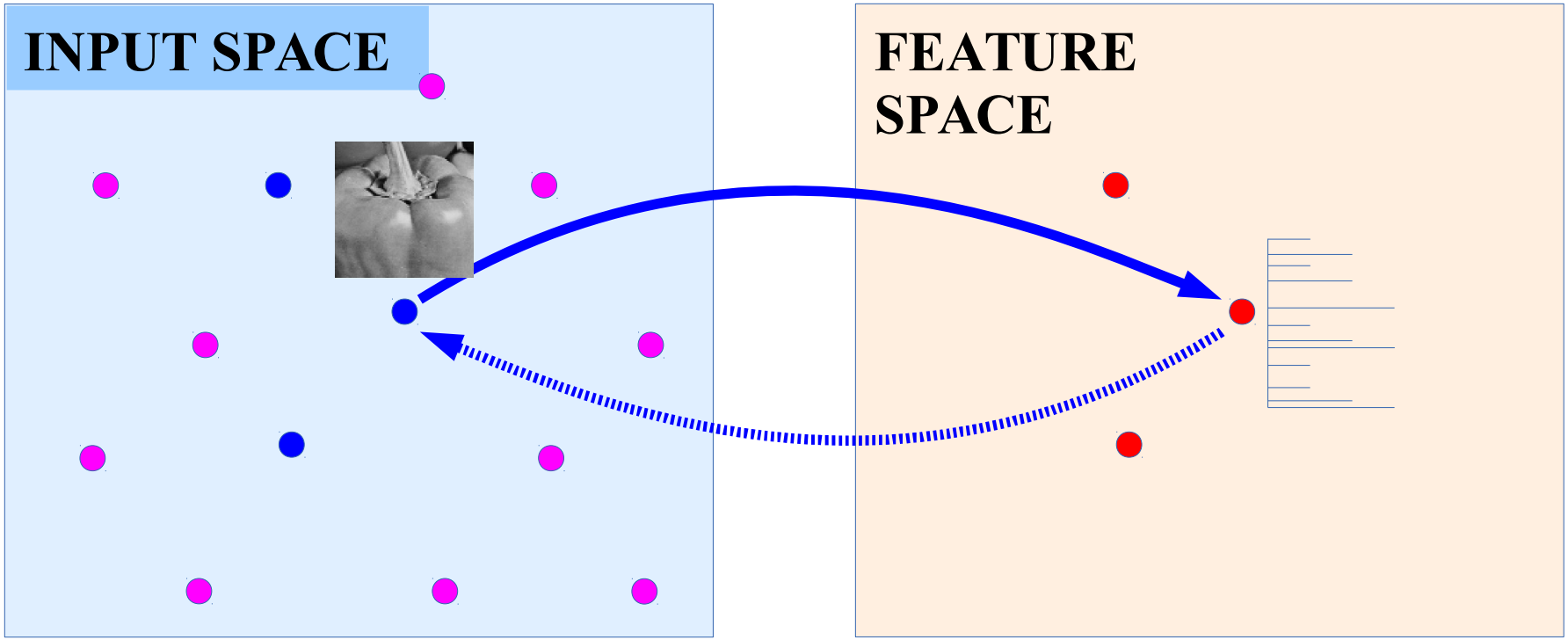




Why Limit the Information Content of the Code?

- Training sample
- Input vector which is NOT a training sample
- Feature vector

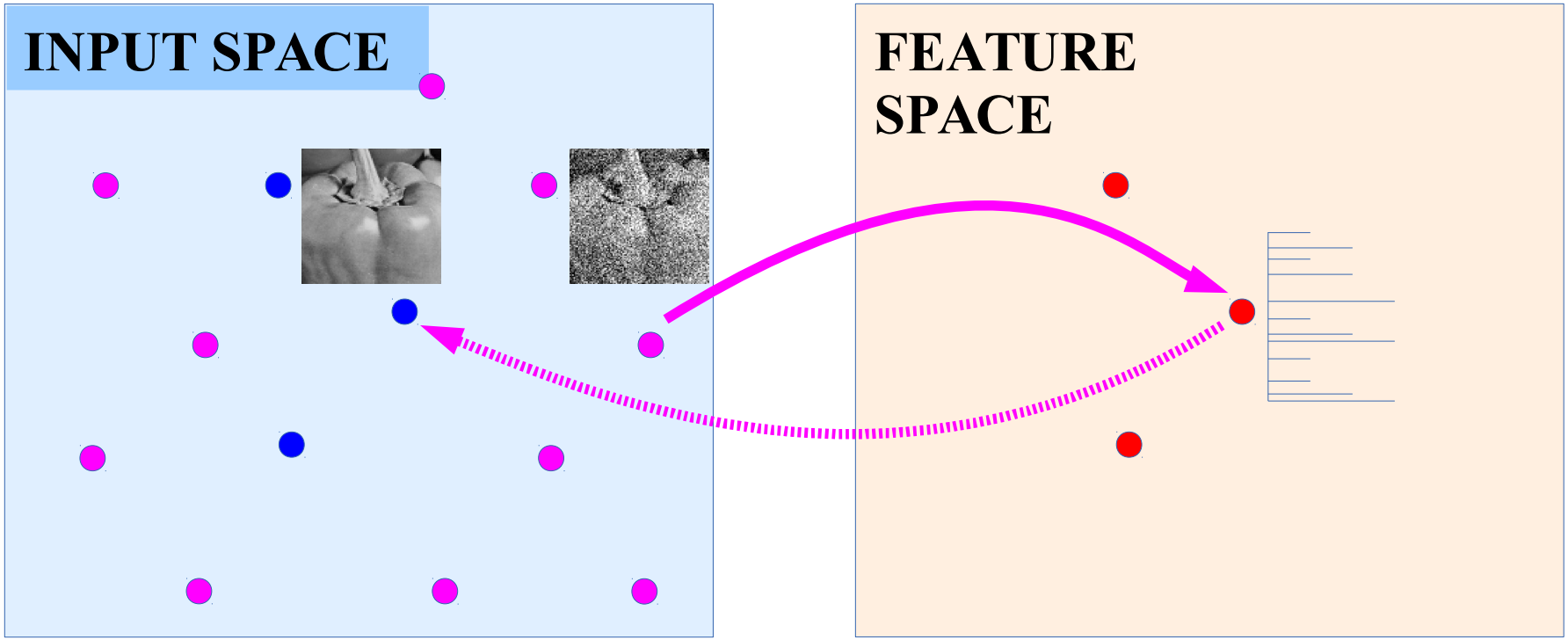
IDEA: reduce number of available codes.



Why Limit the Information Content of the Code?

- Training sample
- Input vector which is NOT a training sample
- Feature vector

IDEA: reduce number of available codes.



Predictive Sparse Decomposition (PSD): sparse auto-encoder

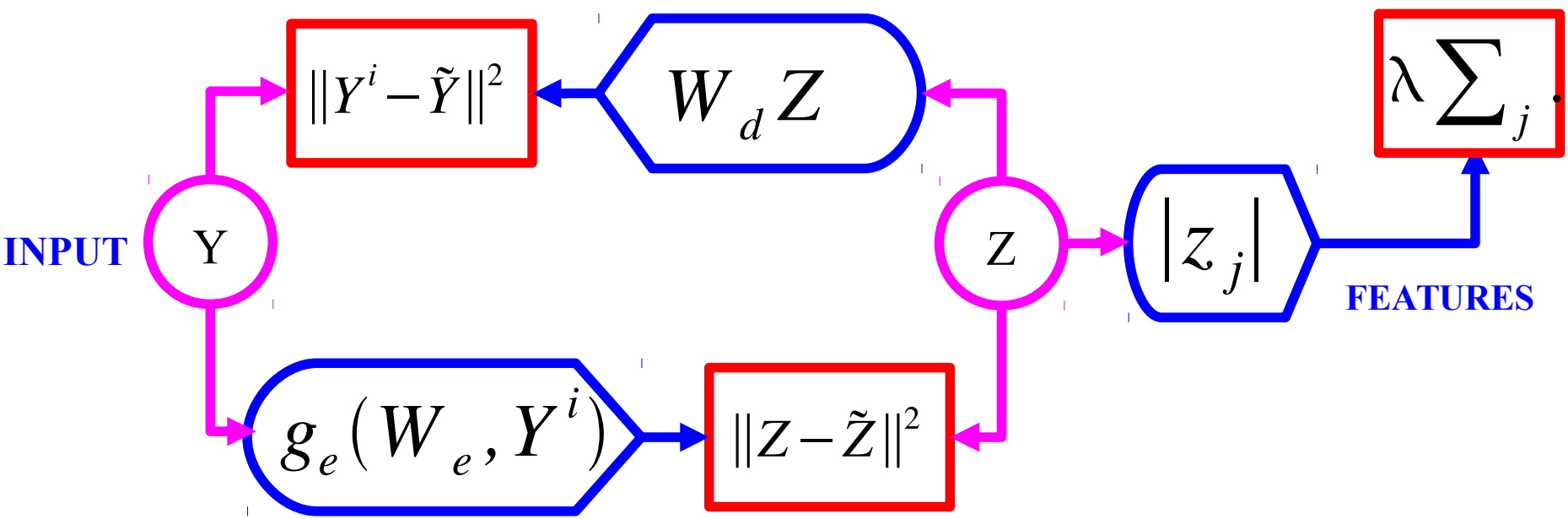
[Kavukcuoglu, Ranzato, LeCun, 2008 → arXiv:1010.3467],

Prediction the optimal code with a trained encoder

Energy = reconstruction_error + code_prediction_error + code_sparsity

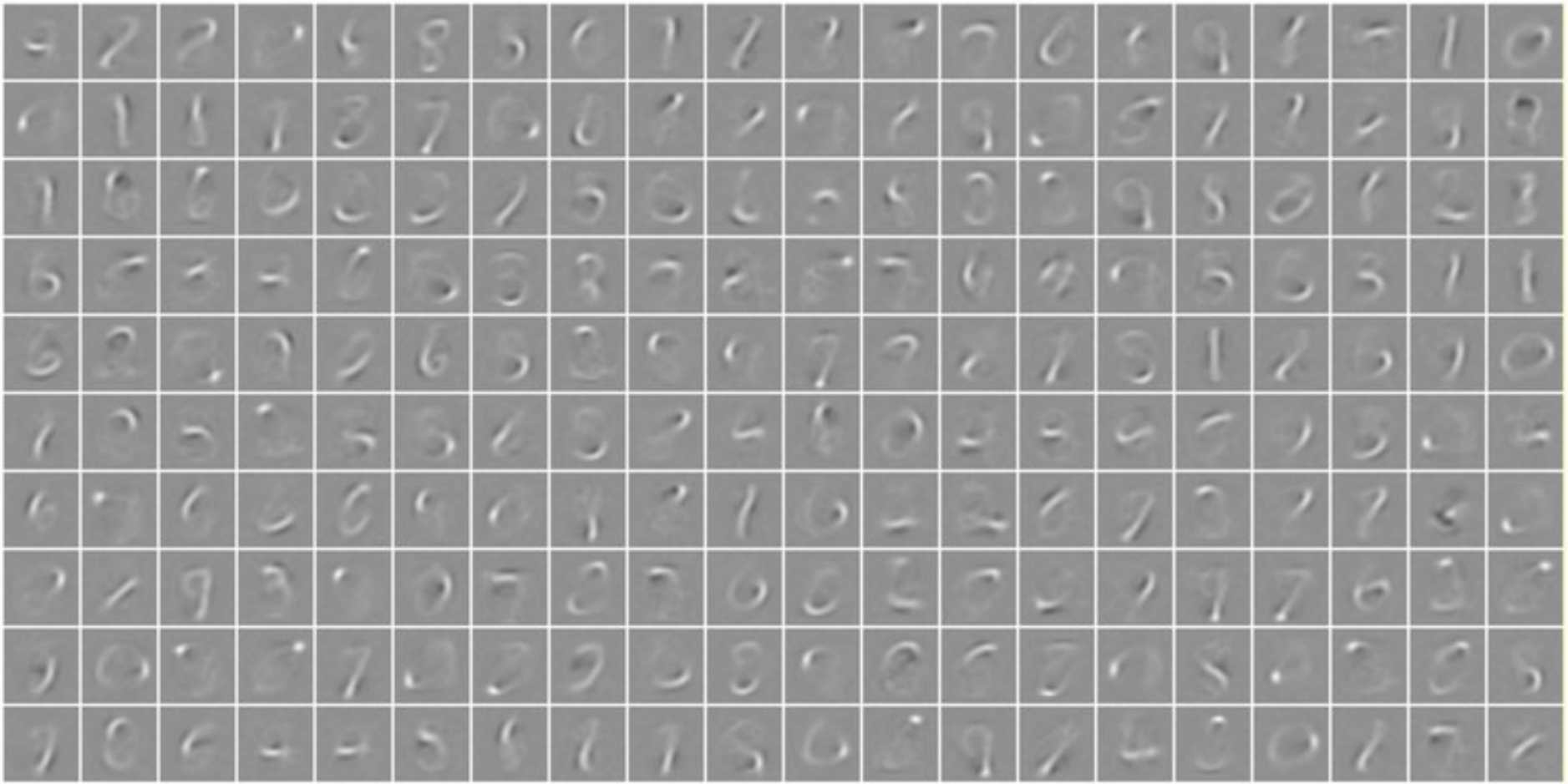
$$E(Y^i, Z) = \|Y^i - W_d Z\|^2 + \|Z - g_e(W_e, Y^i)\|^2 + \lambda \sum_j |z_j|$$

$$g_e(W_e, Y^i) = shrinkage(W_e Y^i)$$



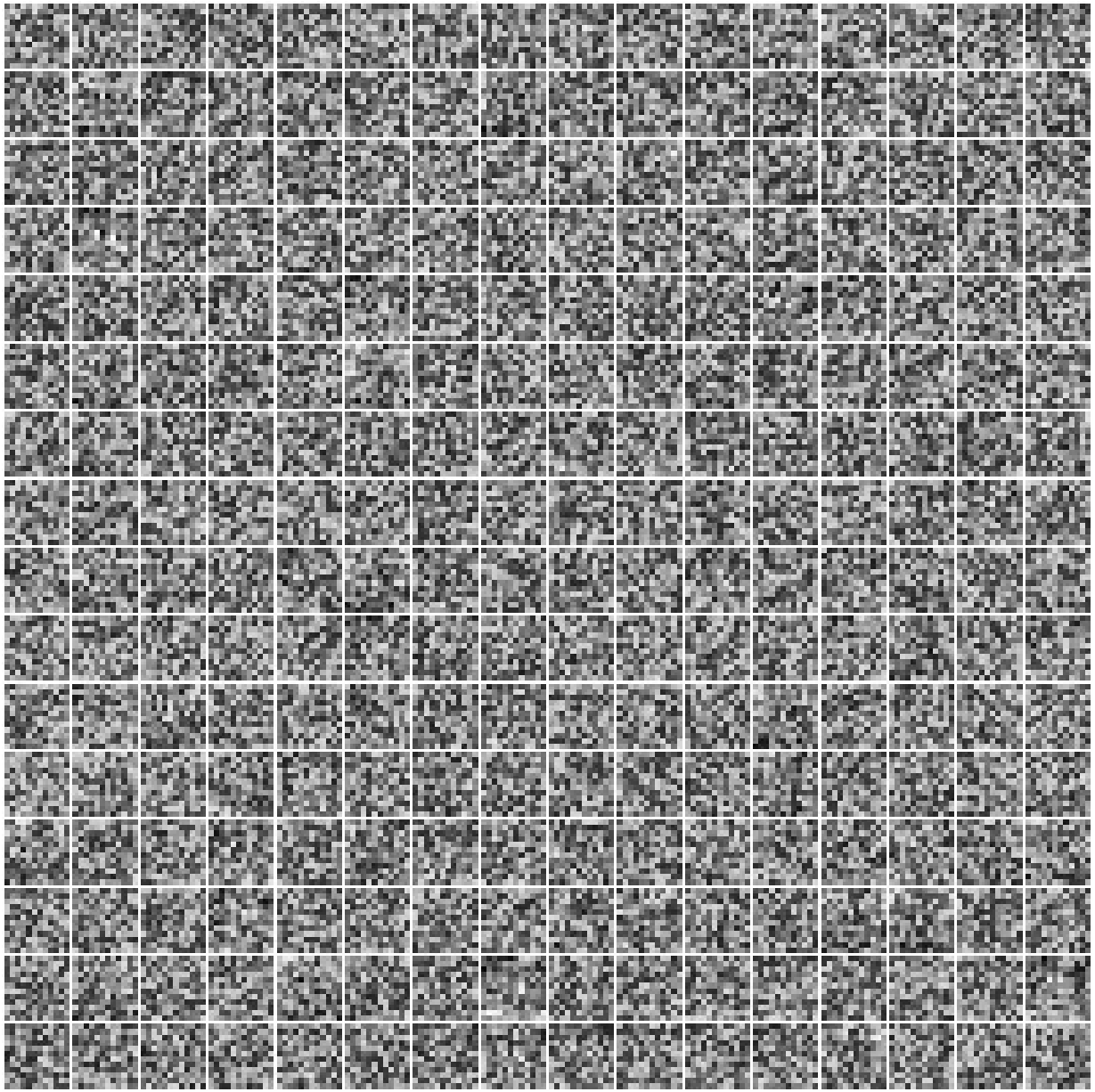
PSD: Basis Functions on MNIST

- Basis functions (and encoder matrix) are digit parts



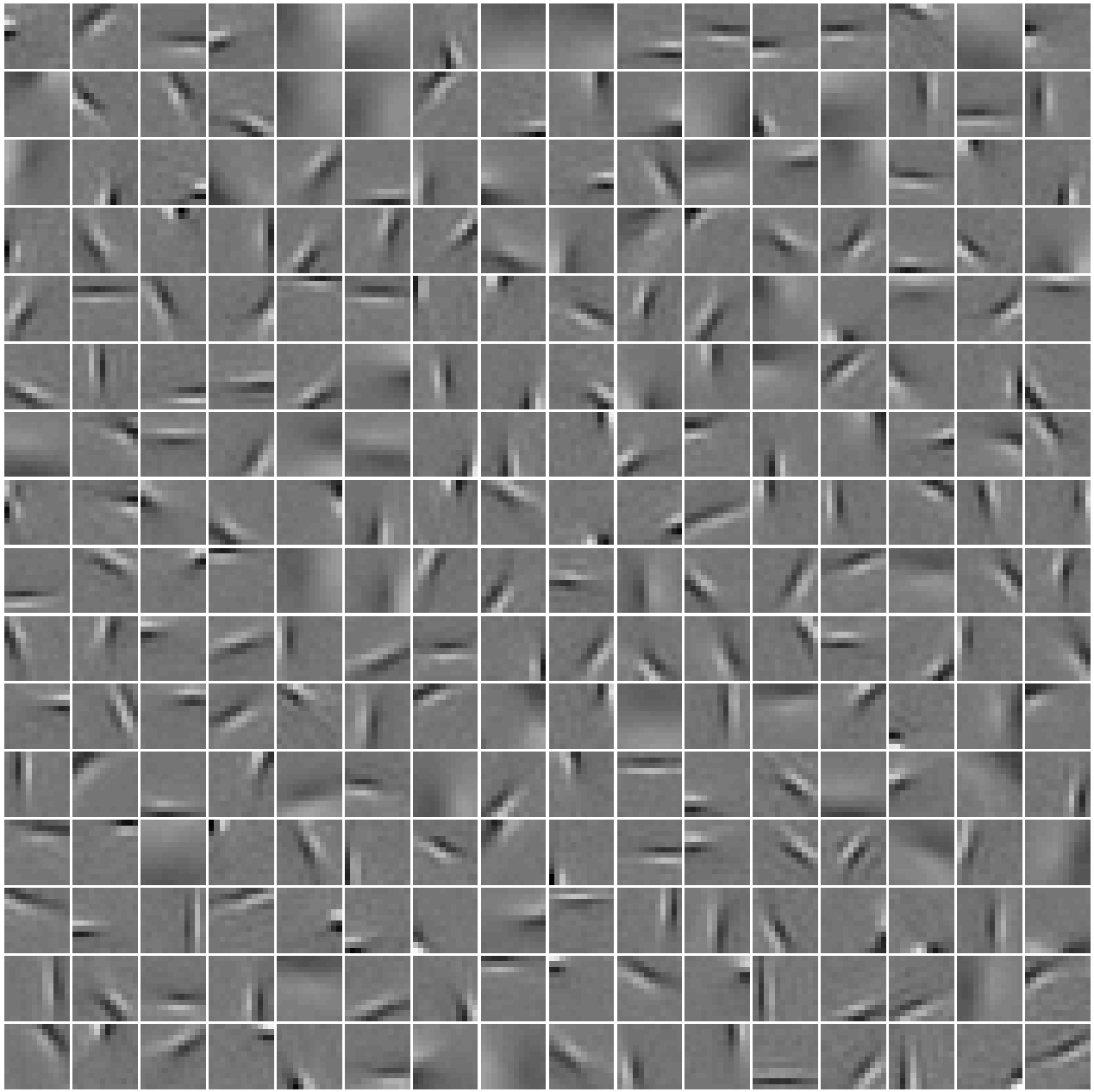
Predictive Sparse Decomposition (PSD): Training

- Training on natural images patches.
  - 12X12
  - 256 basis functions



iteration no 0

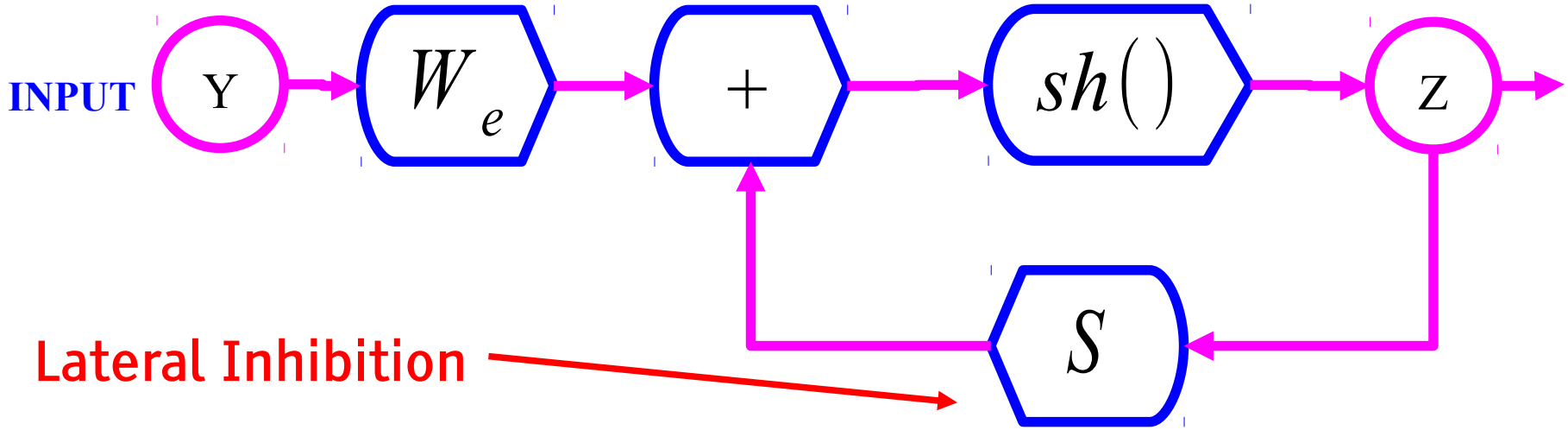
Learned Features on natural patches: V1-like receptive fields



Better Idea: Give the "right" structure to the encoder

- ISTA/FISTA: iterative algorithm that converges to optimal sparse code

[Gregor & LeCun, ICML 2010], [Bronstein et al. ICML 2012], [Rolfe & LeCun ICLR 2013]

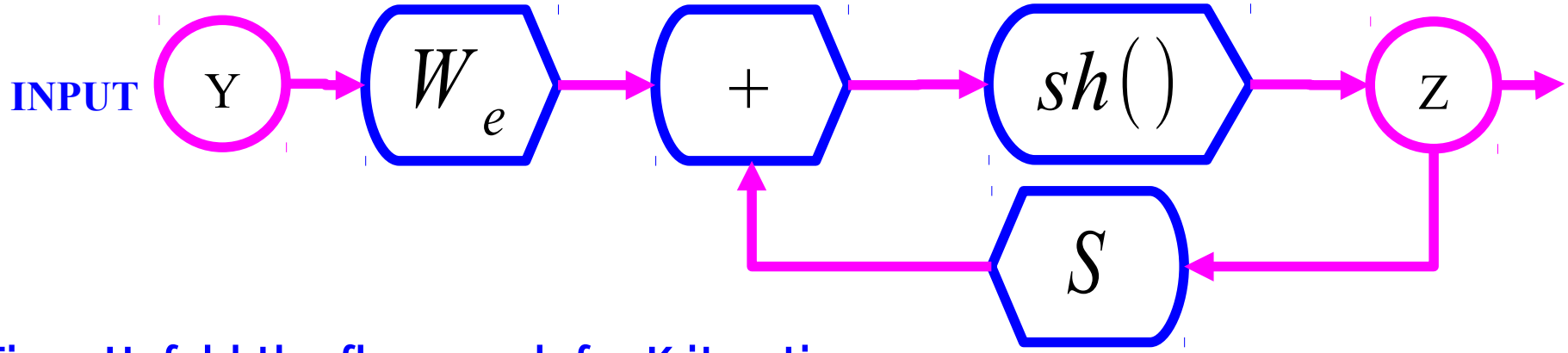


$$Z(t+1) = \text{Shrinkage}_{\lambda/L}\left[Z(t) - \frac{1}{L}W_d^T(W_d Z(t) - Y)\right]$$

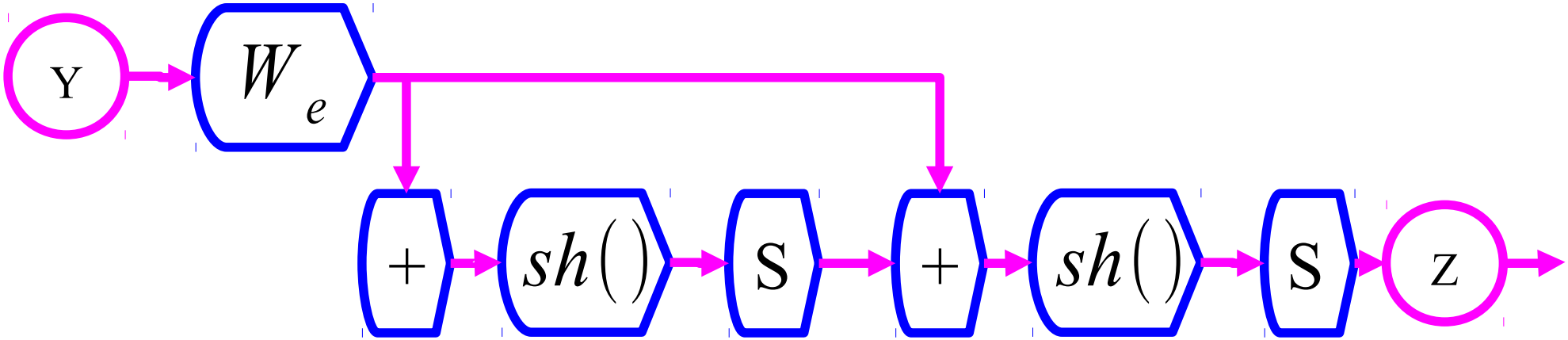
$$Z(t+1) = \text{Shrinkage}_{\lambda/L}\left[W_e^T Y + SZ(t)\right]; \quad W_e = \frac{1}{L}W_d; \quad S = I - \frac{1}{L}W_d^T W_d$$

LISTA: Train We and S matrices to give a good approximation quickly

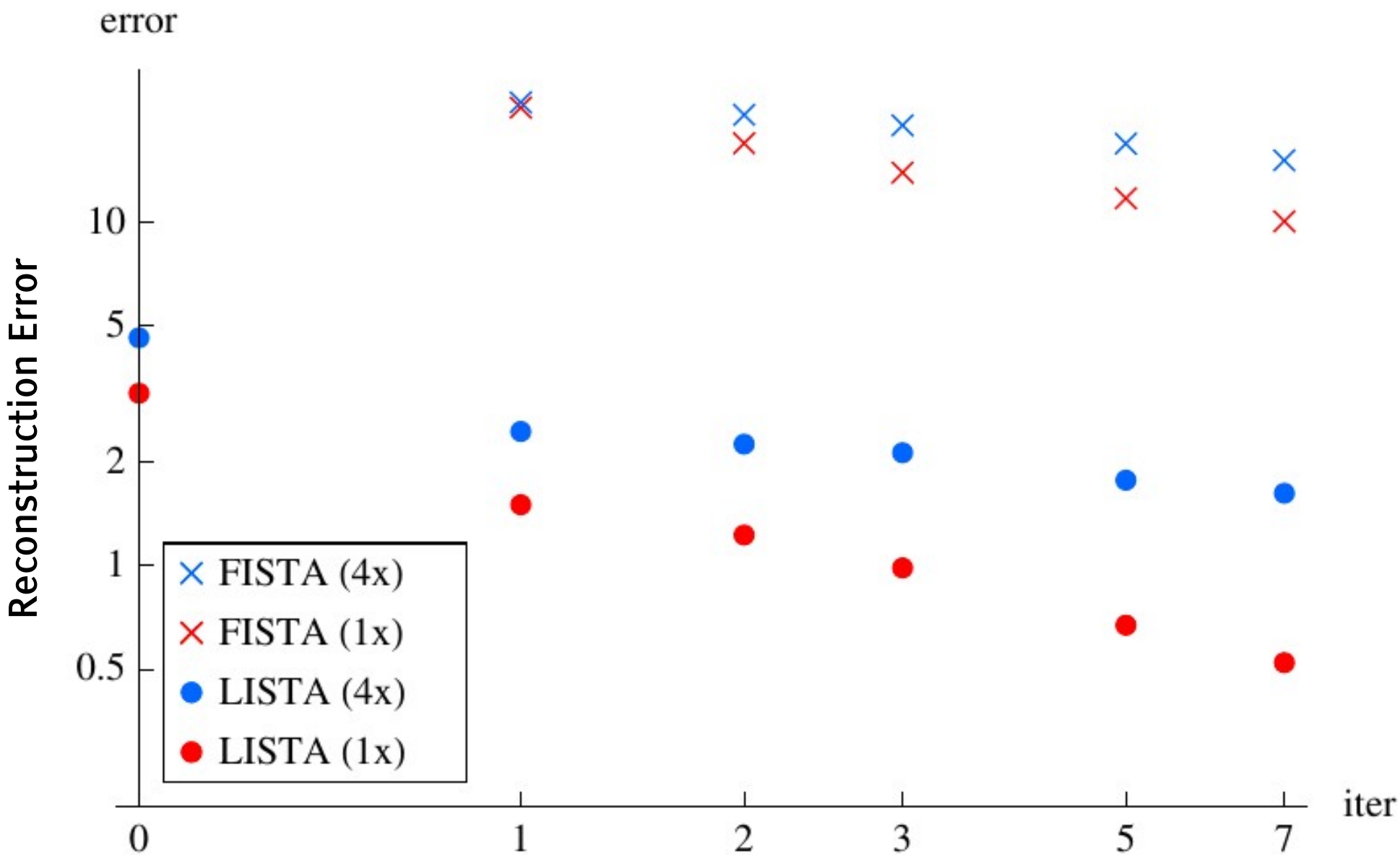
- Think of the FISTA flow graph as a recurrent neural net where We and S are trainable parameters



- Time-Unfold the flow graph for K iterations
- Learn the We and S matrices with "backprop-through-time"
- Get the best approximate solution within K iterations

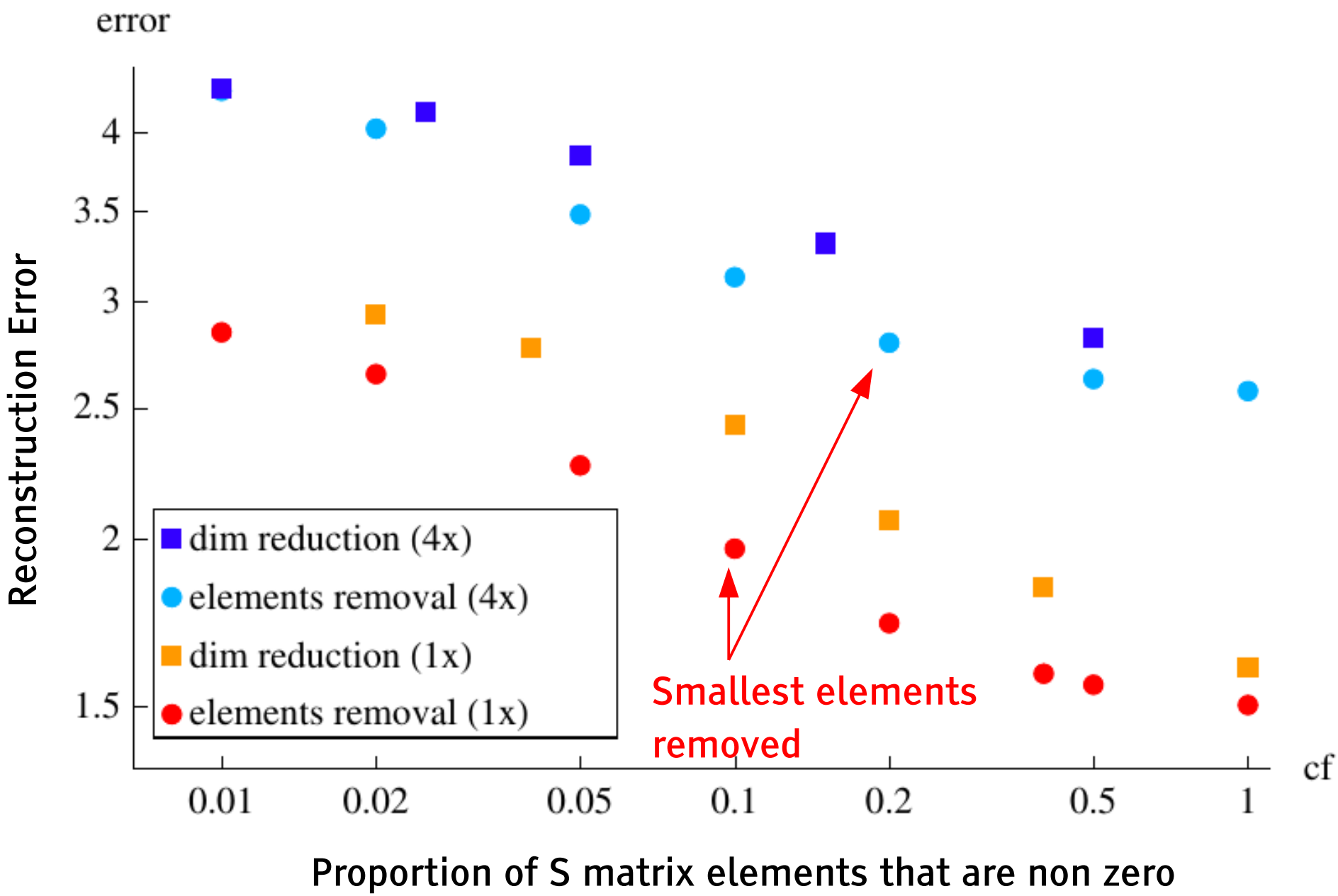


Learning ISTA (LISTA) vs ISTA/FISTA

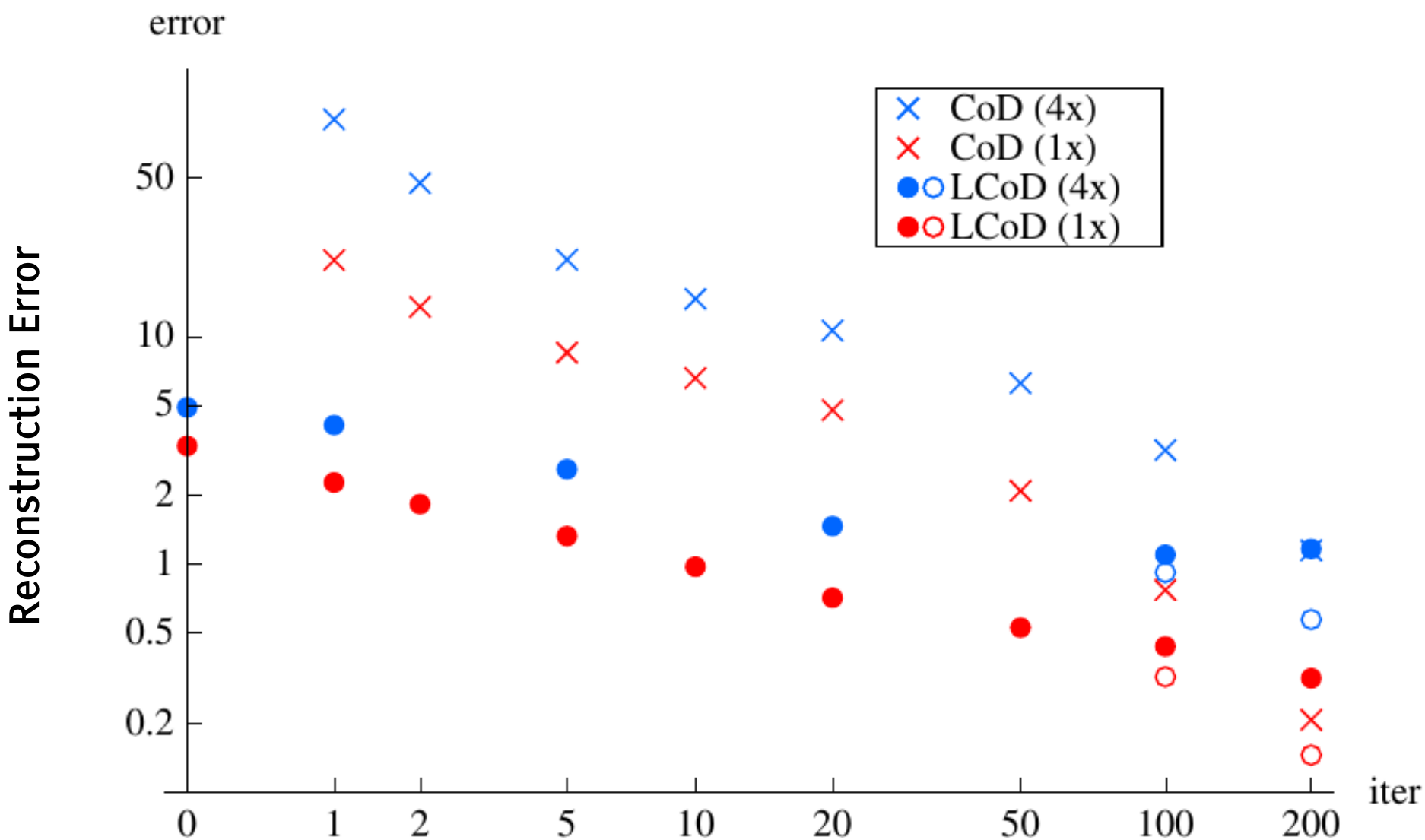


Number of LISTA or FISTA iterations

LISTA with partial mutual inhibition matrix



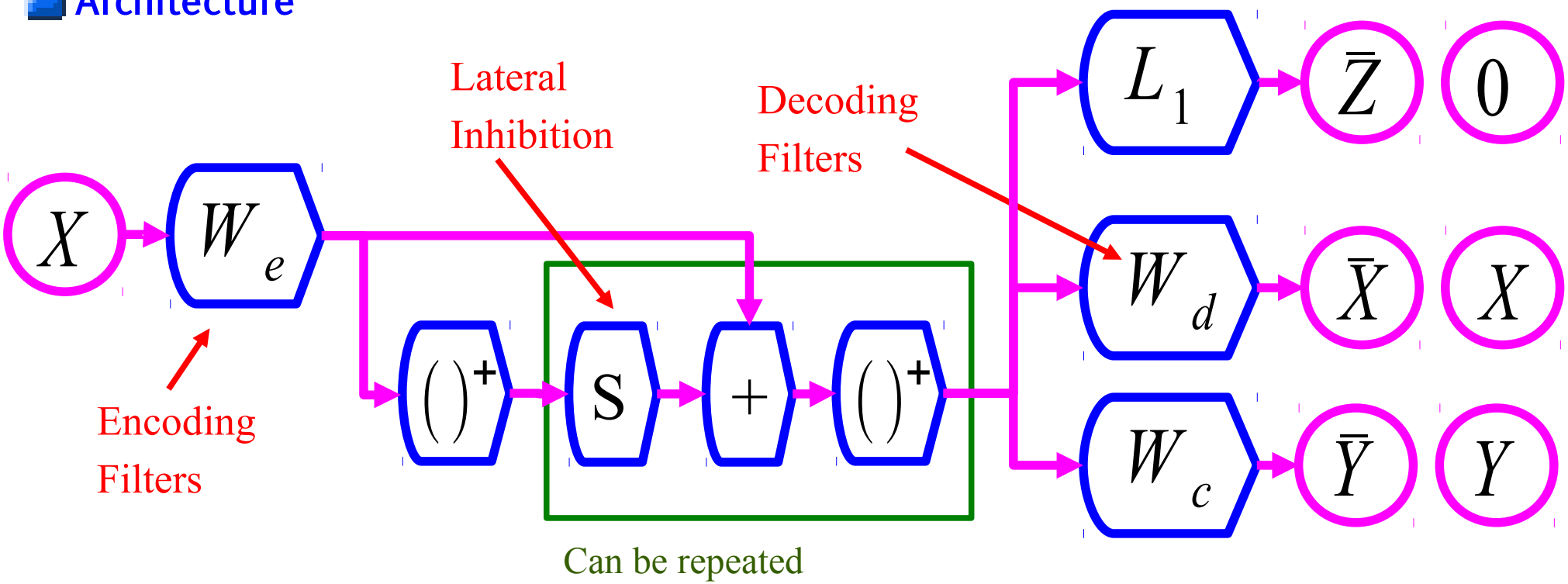
Learning Coordinate Descent (LcoD): faster than LISTA



Number of LISTA or FISTA iterations

Discriminative Recurrent Sparse Auto-Encoder (DrSAE)

- Architecture

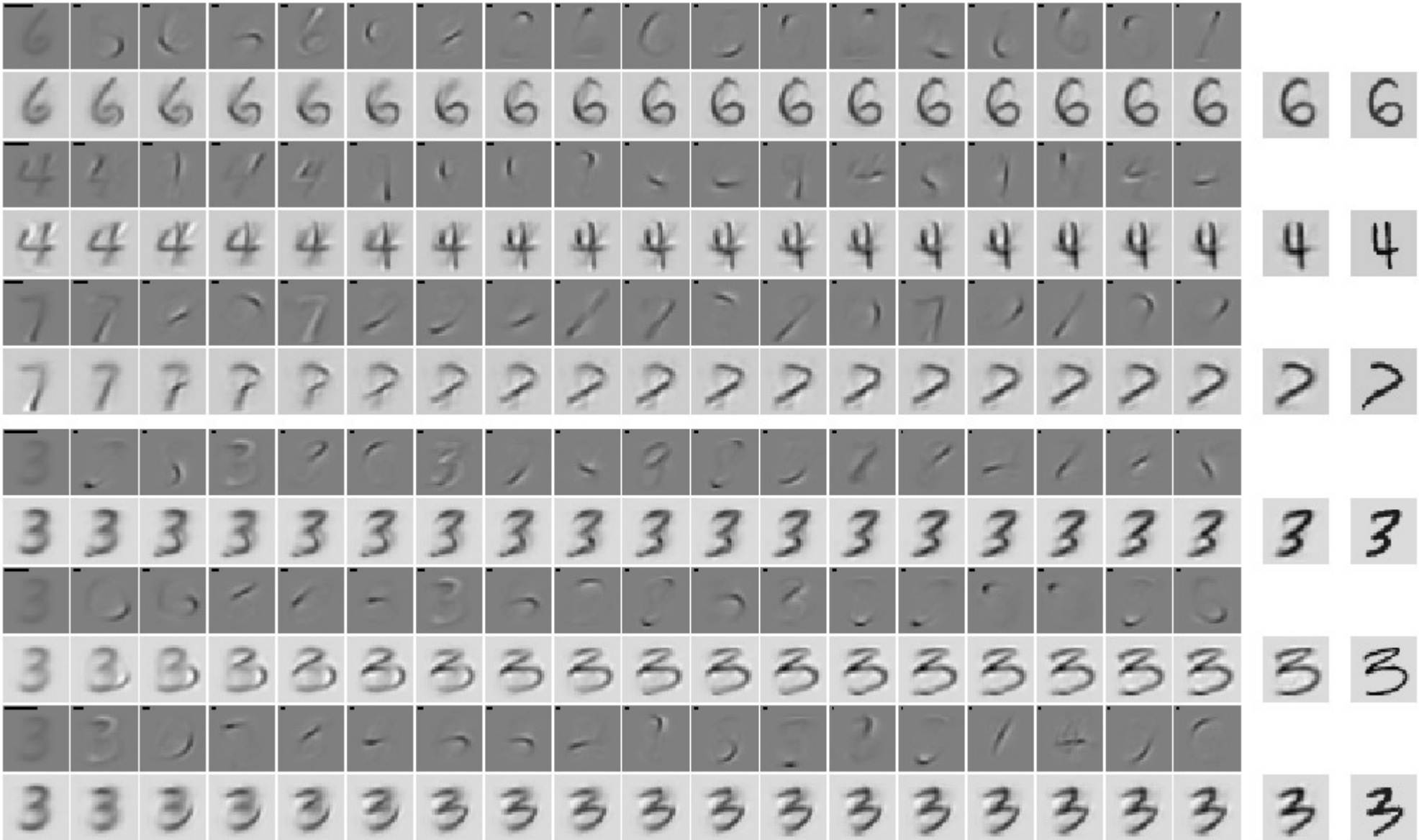


- Rectified linear units
- Classification loss: cross-entropy
- Reconstruction loss: squared error
- Sparsity penalty: L1 norm of last hidden layer
- Rows of Wd and columns of We constrained in unit sphere

[Rolfe & LeCun ICLR 2013]

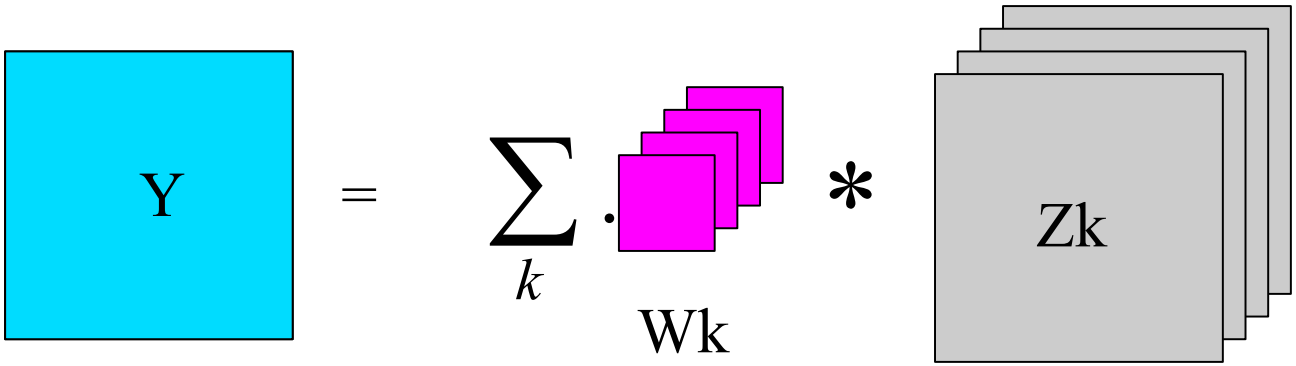
DrSAE Discovers manifold structure of handwritten digits

- Image = prototype + sparse sum of “parts” (to move around the manifold)



Convolutional Sparse Coding

- **Replace the dot products with dictionary element by convolutions.**
  - Input Y is a full image
  - Each code component Zk is a feature map (an image)
  - Each dictionary element is a convolution kernel
- **Regular sparse coding** $$E(Y,Z) = ||Y - \sum_k W_k Z_k||^2 + \alpha \sum_k |Z_k|$$
- **Convolutional S.C.** $$E(Y,Z) = ||Y - \sum_k W_k * Z_k||^2 + \alpha \sum_k |Z_k|$$



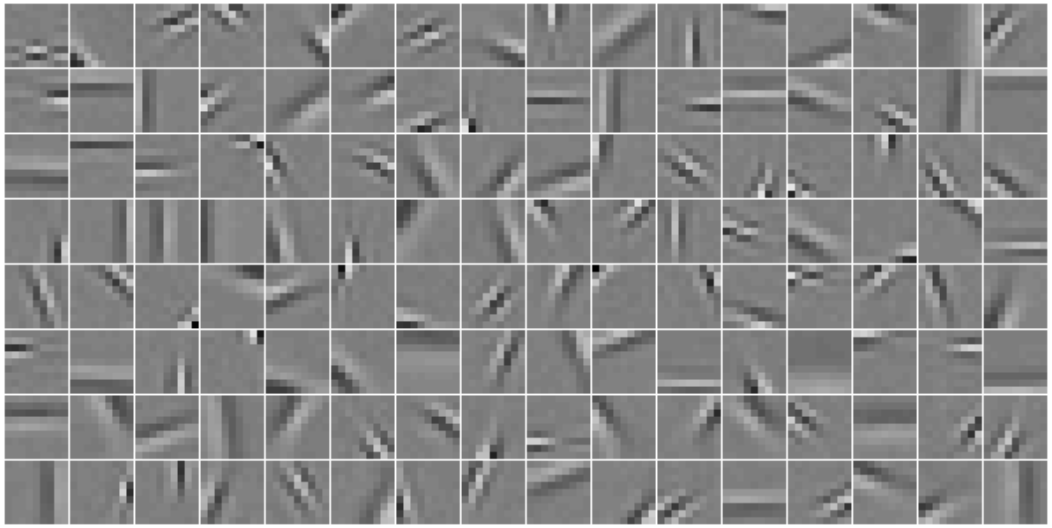
"deconvolutional networks" [Zeiler, Taylor, Fergus CVPR 2010]

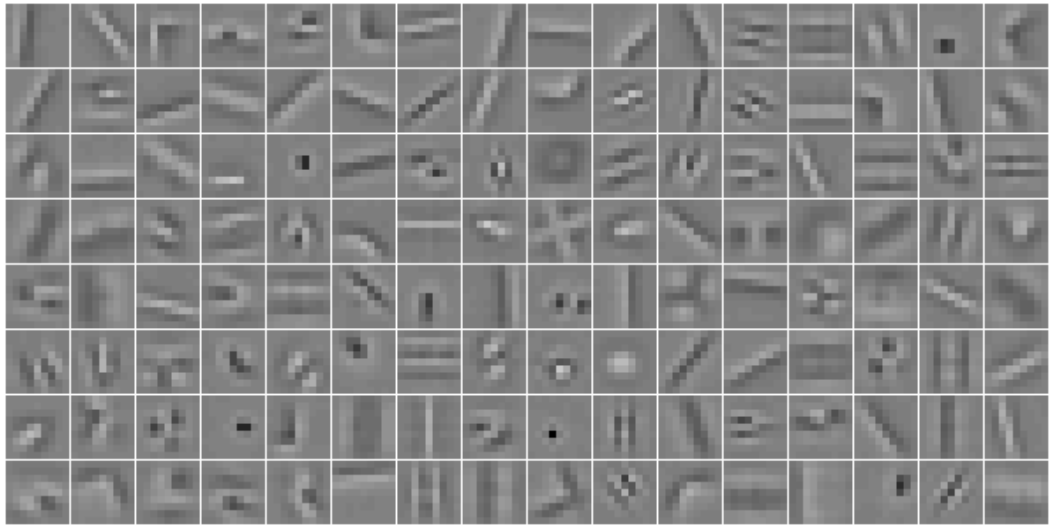
Convolutional PSD: Encoder with a soft sh() Function

Convolutional Formulation

- Extend sparse coding from ***PATCH*** to ***IMAGE***

$$\mathcal{L}(x, z, \mathcal{D}) = \frac{1}{2}||x - \sum_{k=1}^{K} \mathcal{D}_k * z_k||_2^2 + \sum_{k=1}^{K} ||z_k - f(W^k * x)||_2^2 + |z|_1$$

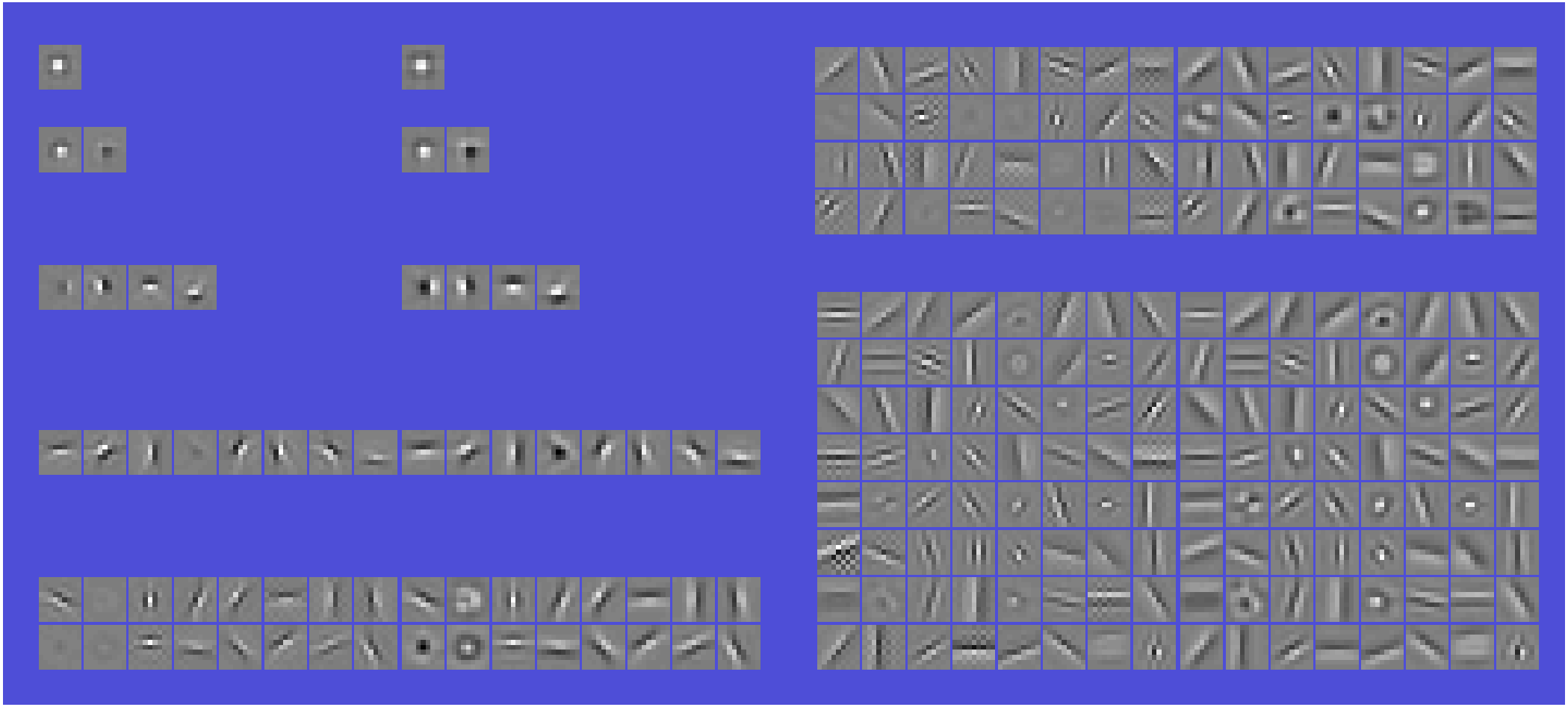




- ***PATCH*** based learning
- ***CONVOLUTIONAL*** learning

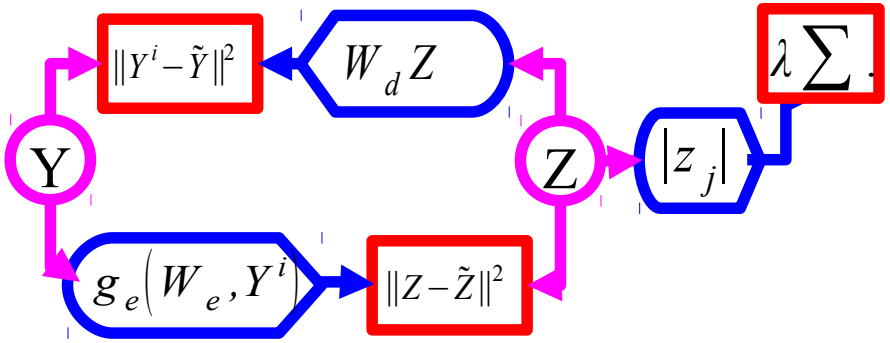
Convolutional Sparse Auto-Encoder on Natural Images

- Filters and Basis Functions obtained with 1, 2, 4, 8, 16, 32, and 64 filters.



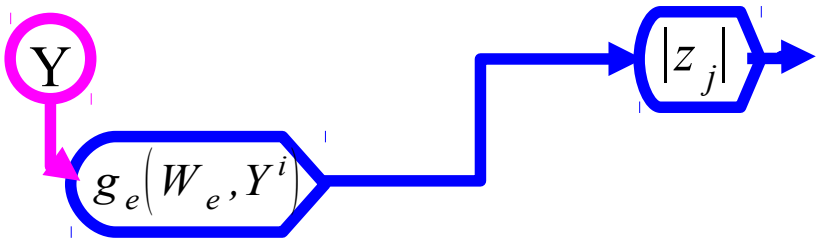
Using PSD to Train a Hierarchy of Features

- Phase 1: train first layer using PSD



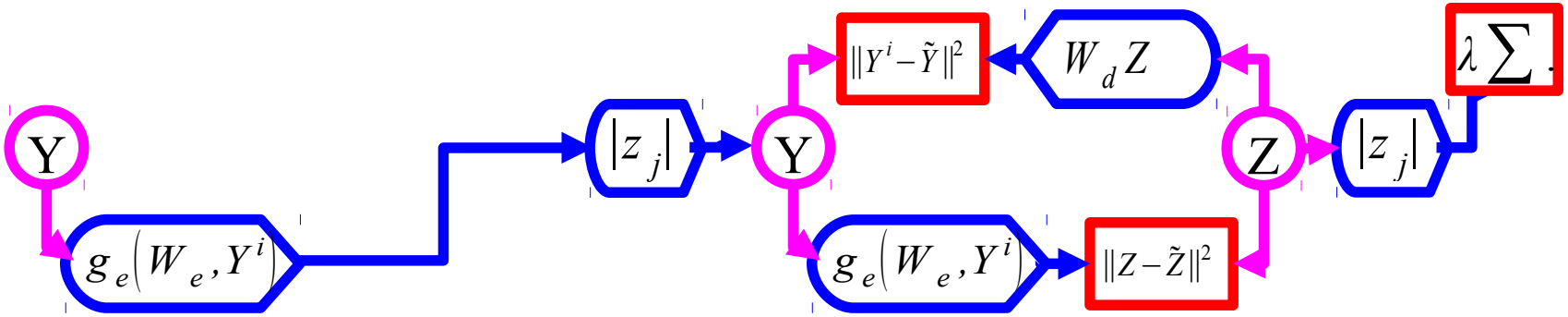
Using PSD to Train a Hierarchy of Features

- Phase 1: train first layer using PSD
- Phase 2: use encoder + absolute value as feature extractor



Using PSD to Train a Hierarchy of Features

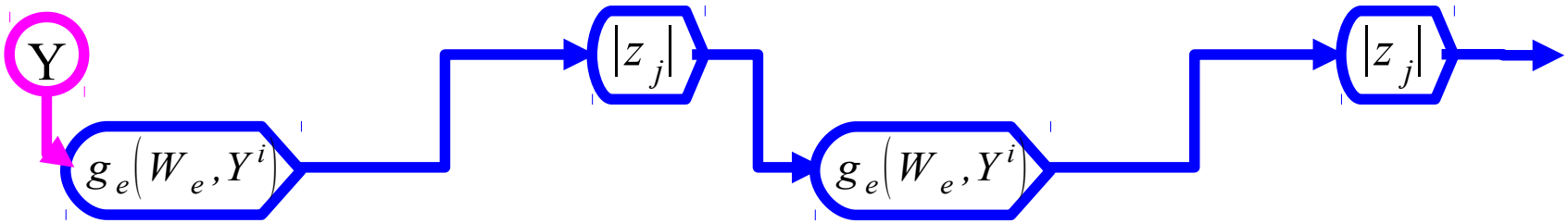
- Phase 1: train first layer using PSD
- Phase 2: use encoder + absolute value as feature extractor
- Phase 3: train the second layer using PSD



FEATURES

Using PSD to Train a Hierarchy of Features

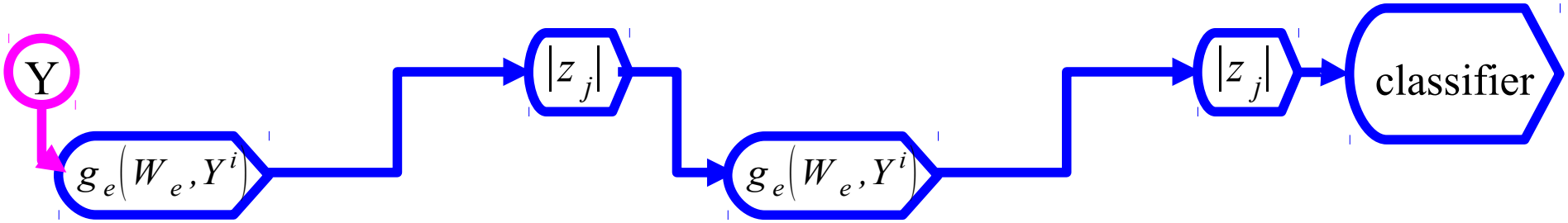
- Phase 1: train first layer using PSD
- Phase 2: use encoder + absolute value as feature extractor
- Phase 3: train the second layer using PSD
- Phase 4: use encoder + absolute value as 2nd feature extractor



FEATURES

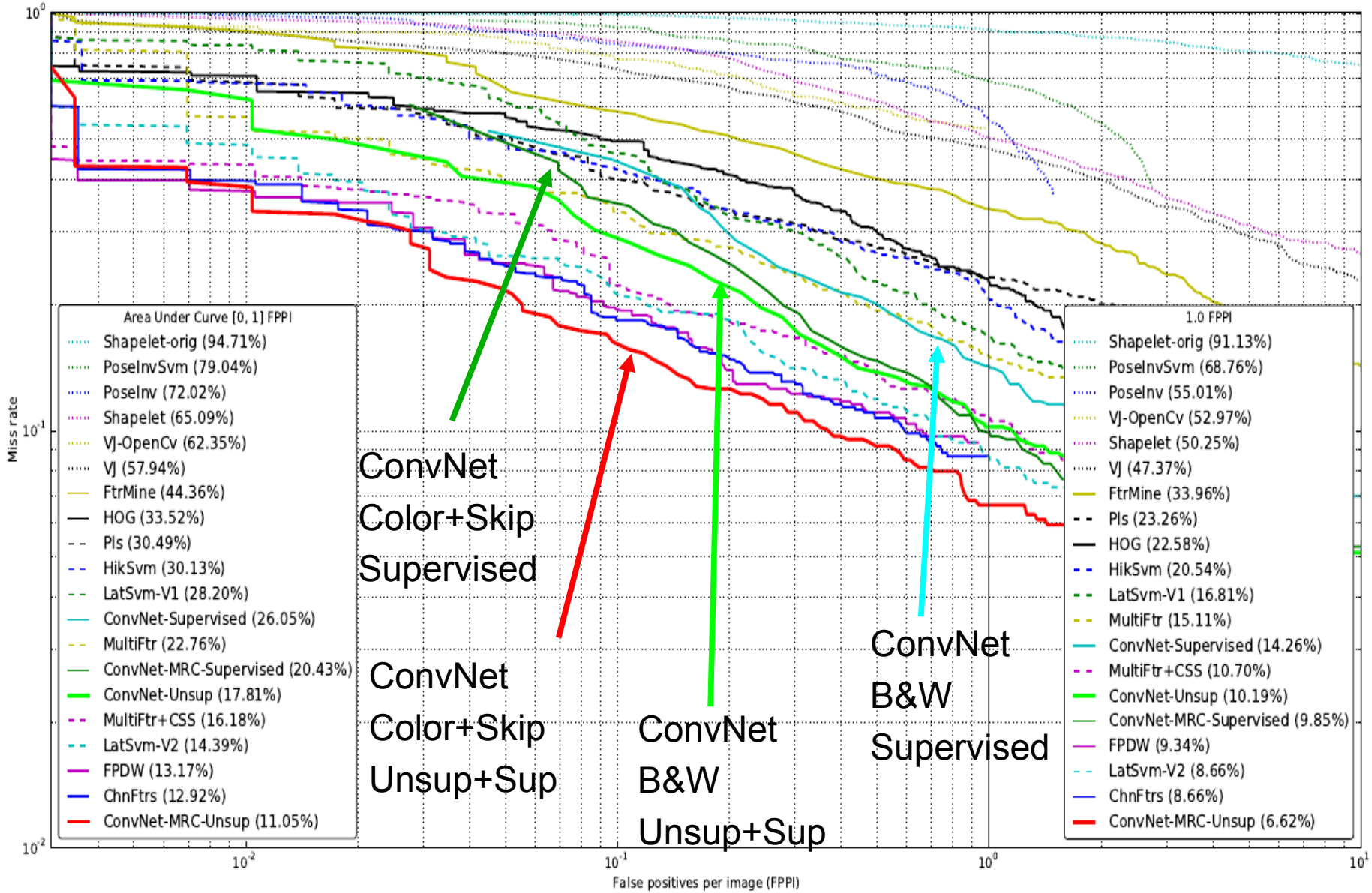
Using PSD to Train a Hierarchy of Features

- Phase 1: train first layer using PSD
- Phase 2: use encoder + absolute value as feature extractor
- Phase 3: train the second layer using PSD
- Phase 4: use encoder + absolute value as 2nd feature extractor
- Phase 5: train a supervised classifier on top
- Phase 6 (optional): train the entire system with supervised back-propagation



FEATURES

Pedestrian Detection: INRIA Dataset. Miss rate vs false positives



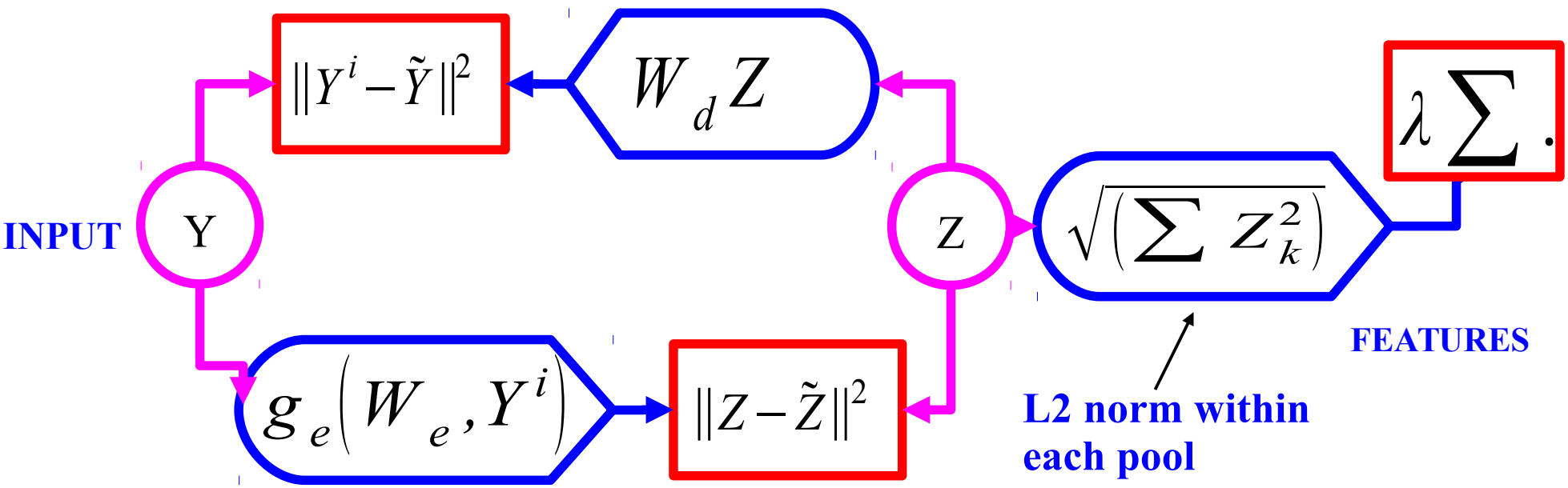
[Kavukcuoglu et al. NIPS 2010] [Sermanet et al. ArXiv 2012]

Unsupervised Learning: Invariant Features

Learning Invariant Features with L2 Group Sparsity

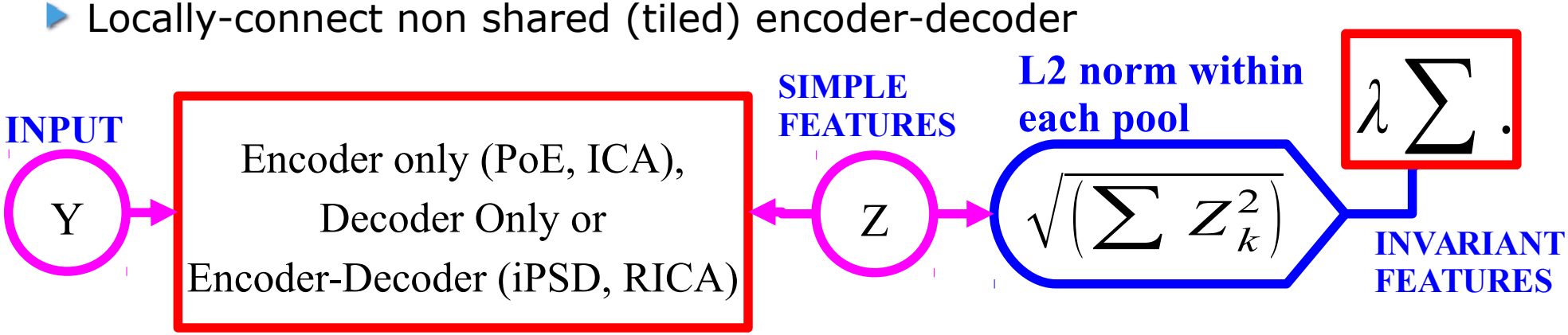
- Unsupervised PSD ignores the spatial pooling step.
- Could we devise a similar method that learns the pooling layer as well?
- Idea [Hyvarinen & Hoyer 2001]: group sparsity on pools of features
  - Minimum number of pools must be non-zero
  - Number of features that are on within a pool doesn't matter
  - Pools tend to regroup similar features

$$E(Y,Z)=\|Y-W_d Z\|^2+\|Z-g_e(W_e,Y)\|^2+\sum_j \sqrt{\sum_{k\in P_j} Z_k^2}$$



Learning Invariant Features with L2 Group Sparsity

- Idea: features are pooled in group.
  - Sparsity: sum over groups of L2 norm of activity in group.
- [Hyvärinen Hoyer 2001]: "subspace ICA"
  - decoder only, square
- [Welling, Hinton, Osindero NIPS 2002]: pooled product of experts
  - encoder only, overcomplete, log student-T penalty on L2 pooling
- [Kavukcuoglu, Ranzato, Fergus LeCun, CVPR 2010]: Invariant PSD
  - encoder-decoder (like PSD), overcomplete, L2 pooling
- [Le et al. NIPS 2011]: Reconstruction ICA
  - Same as [Kavukcuoglu 2010] with linear encoder and tied decoder
- [Gregor & LeCun arXiv:1006:0448, 2010] [Le et al. ICML 2012]
  - Locally-connect non shared (tiled) encoder-decoder



Groups are local in a 2D Topographic Map

- The filters arrange themselves spontaneously so that similar filters enter the same pool.
- The pooling units can be seen as complex cells
- Outputs of pooling units are invariant to local transformations of the input
  - For some it's translations, for others rotations, or other transformations.

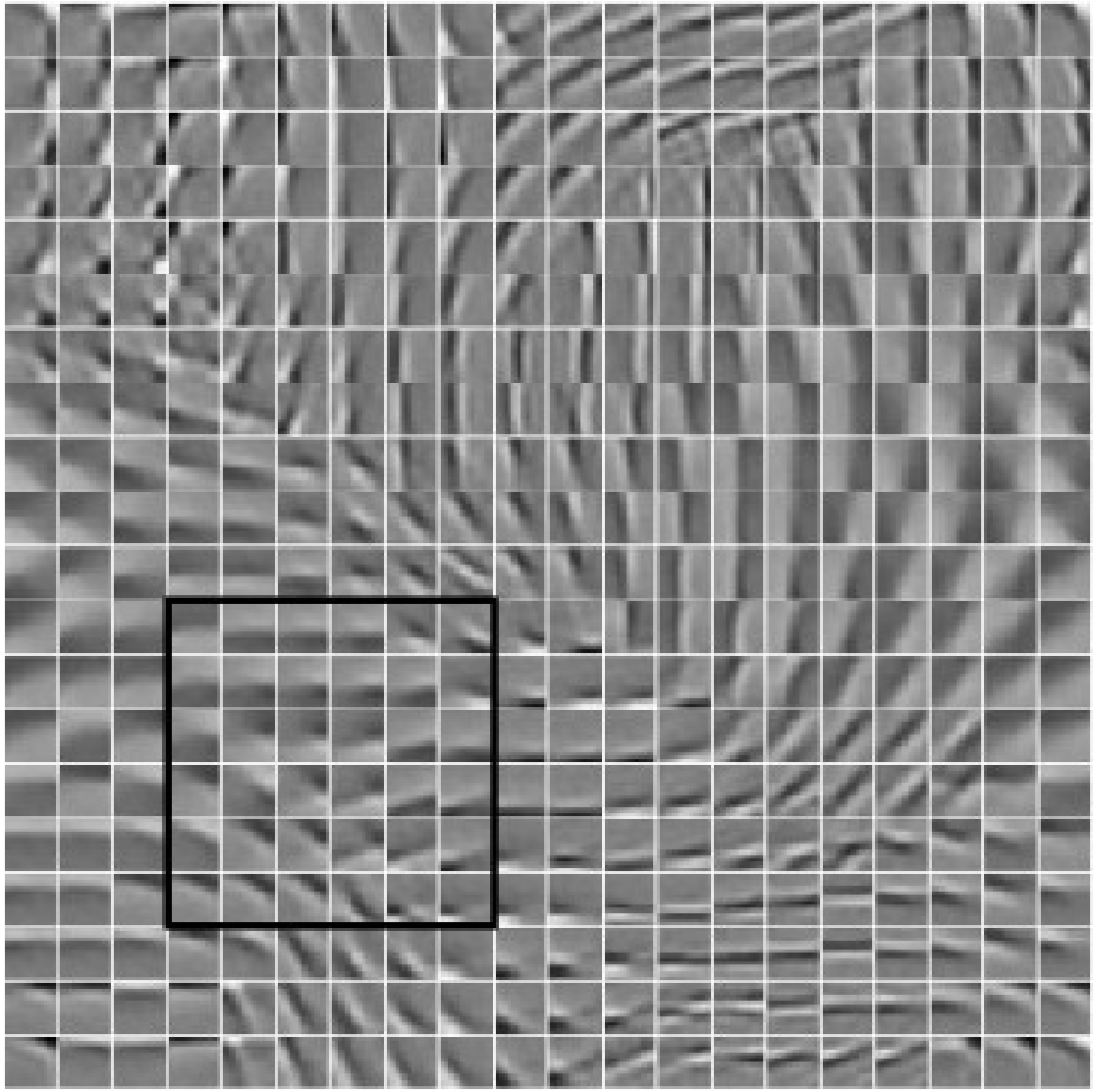


Image-level training, local filters but no weight sharing

- Training on 115x115 images. Kernels are 15x15 (not shared across space!)
  - [Gregor & LeCun 2010]
  - Local receptive fields
  - No shared weights
  - 4x overcomplete
  - L2 pooling
  - Group sparsity over pools

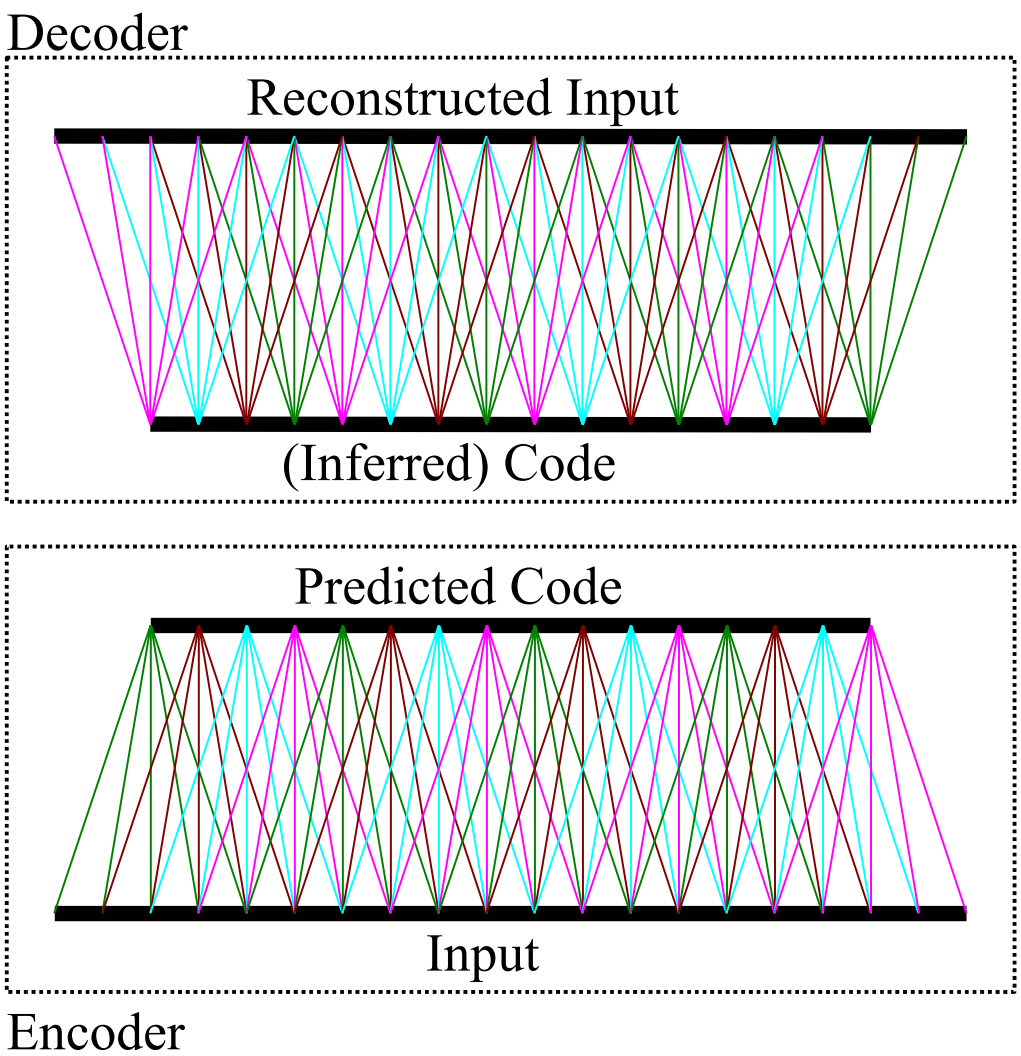
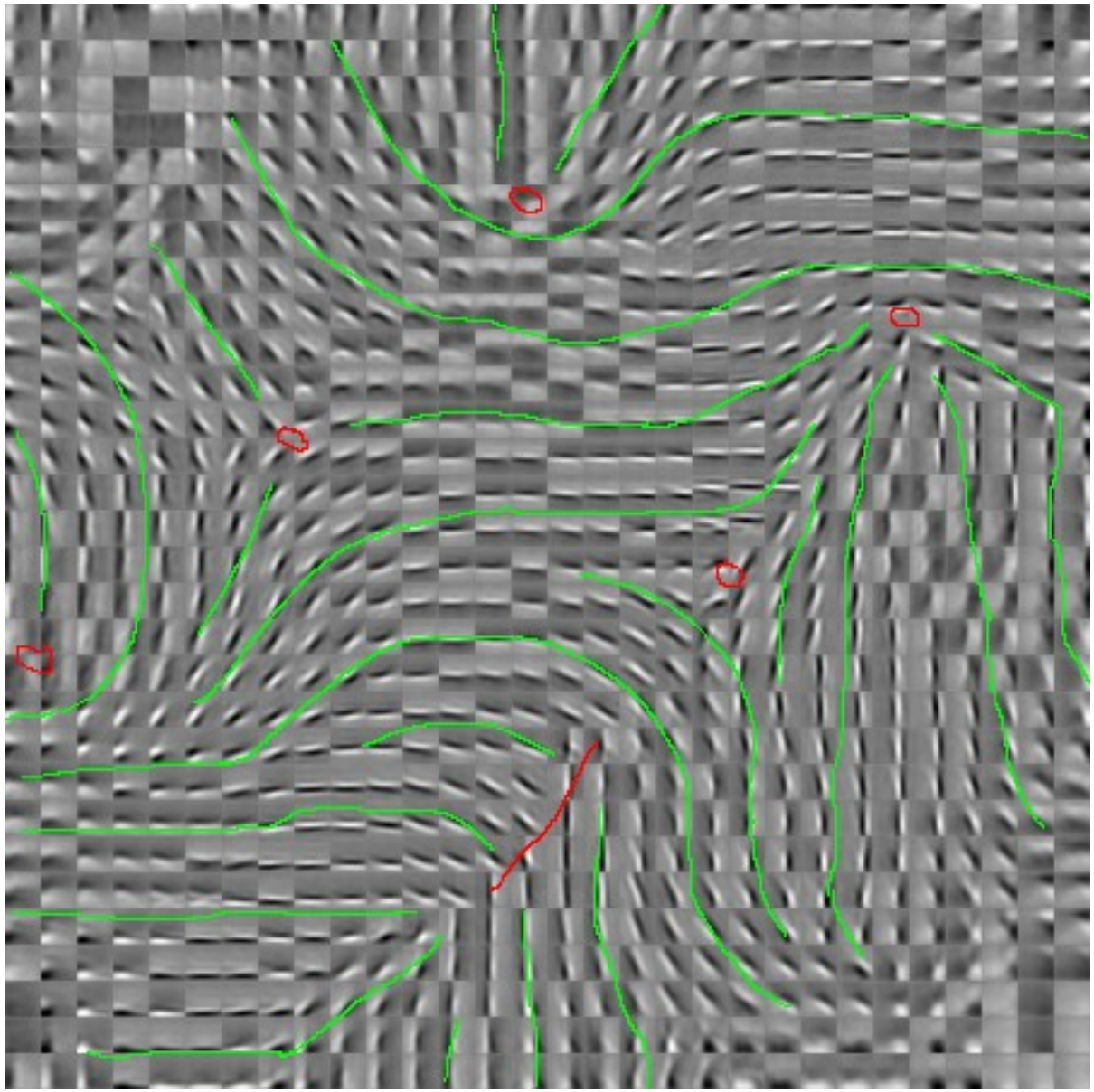


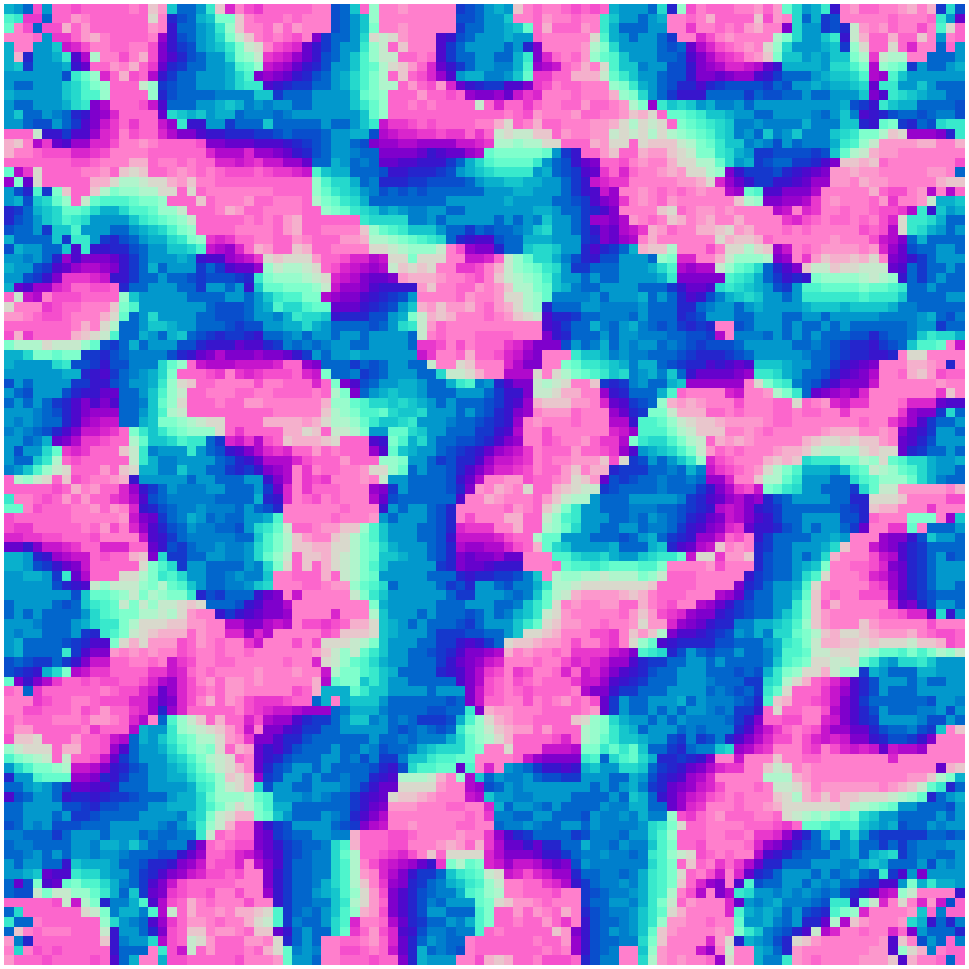
Image-level training, local filters but no weight sharing

- Training on 115x115 images. Kernels are 15x15 (not shared across space!)



Topographic Maps

K Obermayer and GG Blasdel, Journal of Neuroscience, Vol 13, 4114-4129 (**Monkey**)

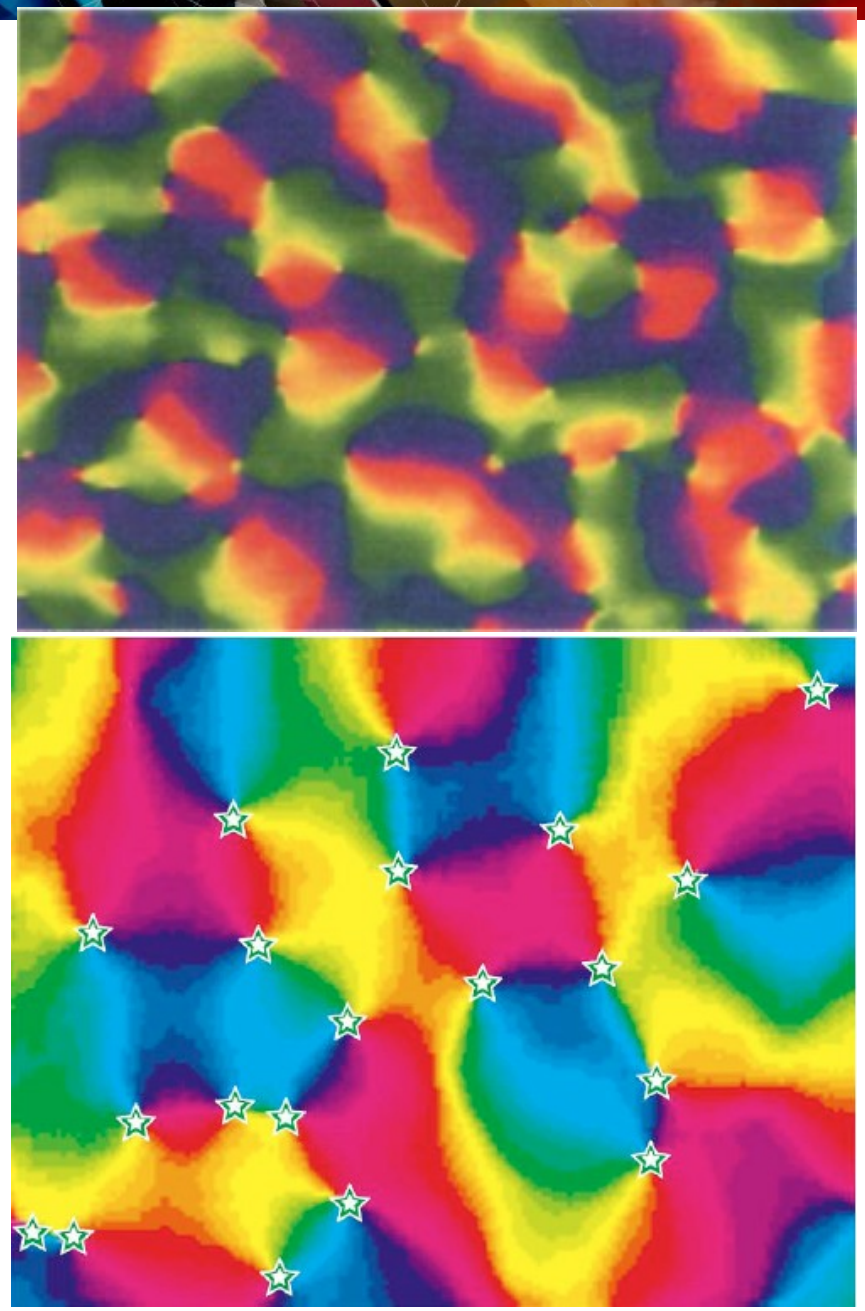


119x119 Image Input

100x100 Code

20x20 Receptive field size

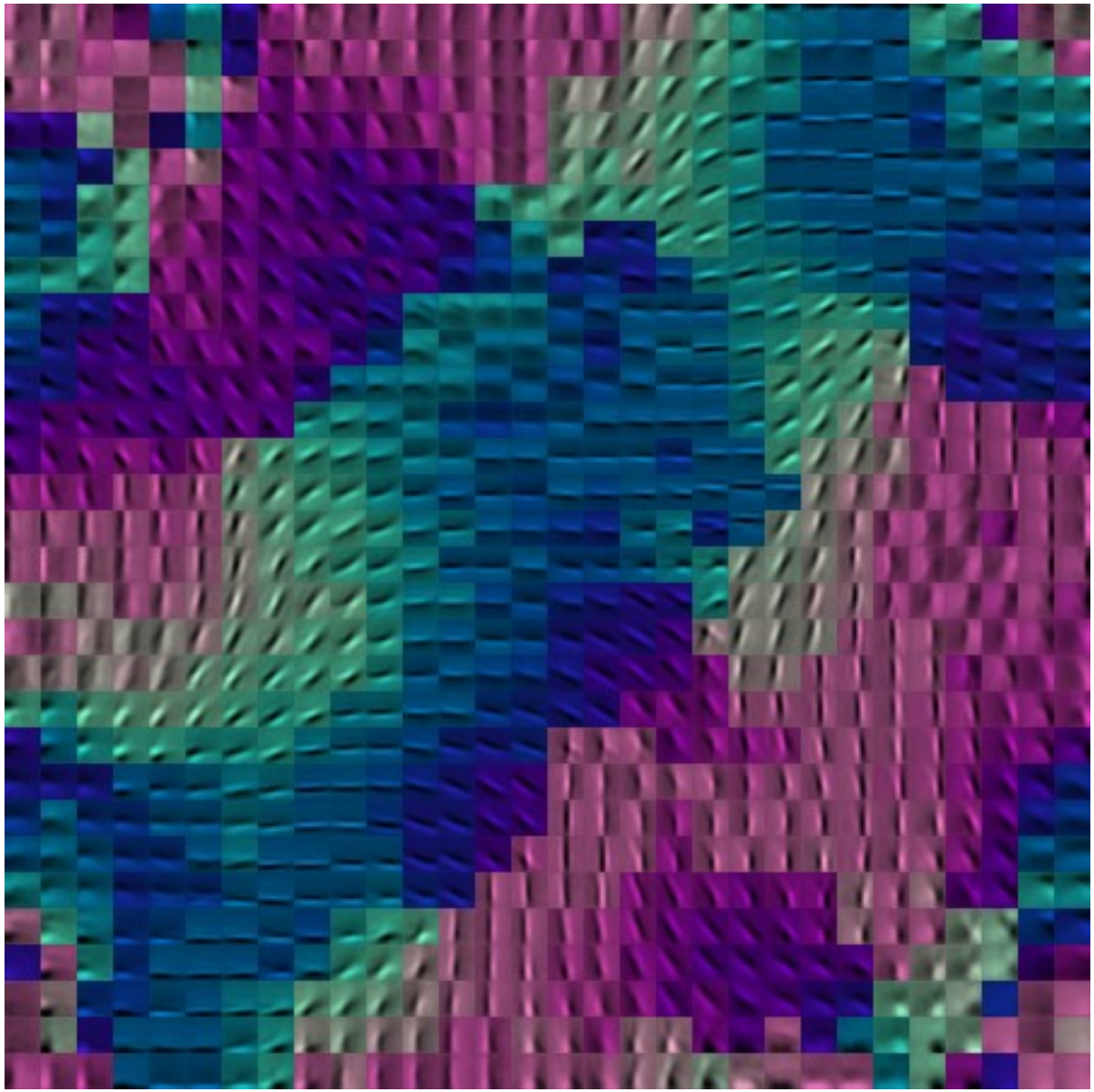
sigma=5



Michael C. Crair, et. al. The Journal of Neurophysiology Vol. 77 No. 6 June 1997, pp. 3381-3385 (**Cat**)

Image-level training, local filters but no weight sharing

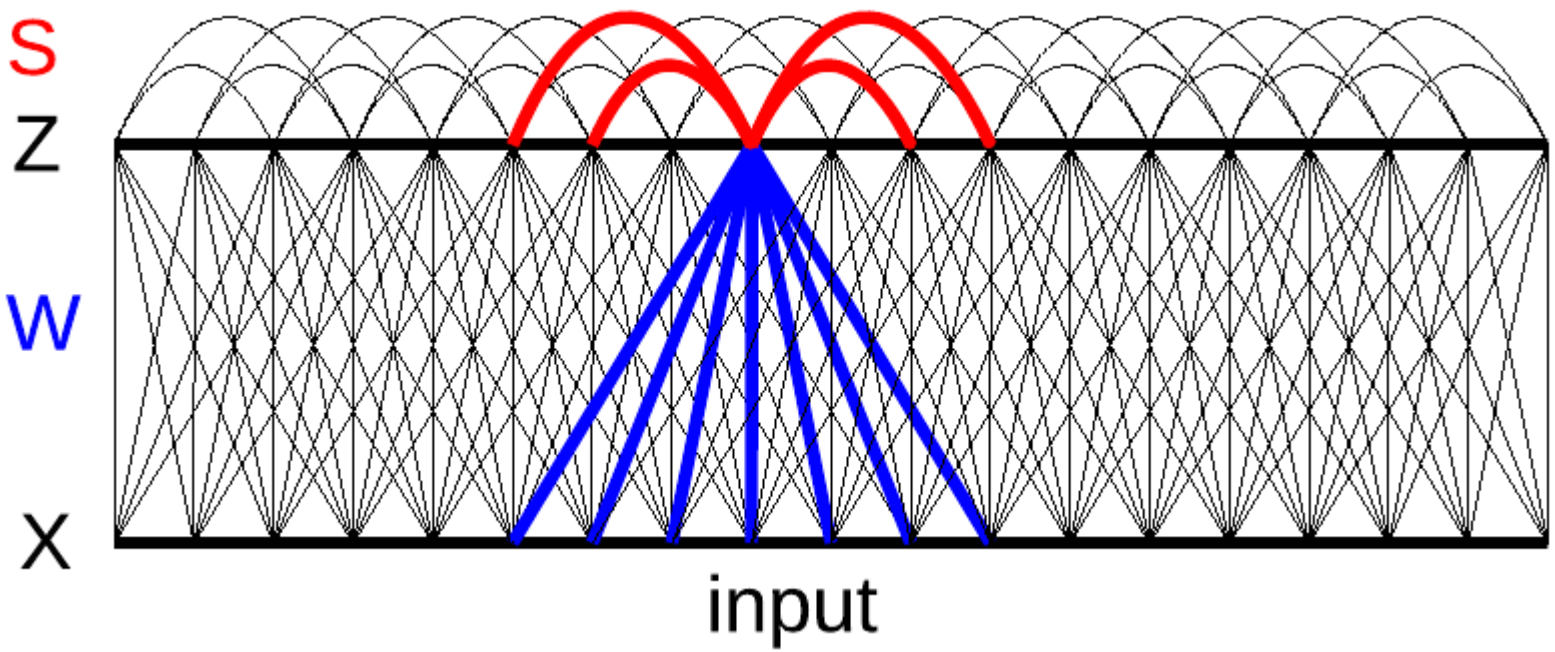
- Color indicates orientation (by fitting Gabors)



Invariant Features Lateral Inhibition

- Replace the L1 sparsity term by a lateral inhibition matrix
- Easy way to impose some structure on the sparsity

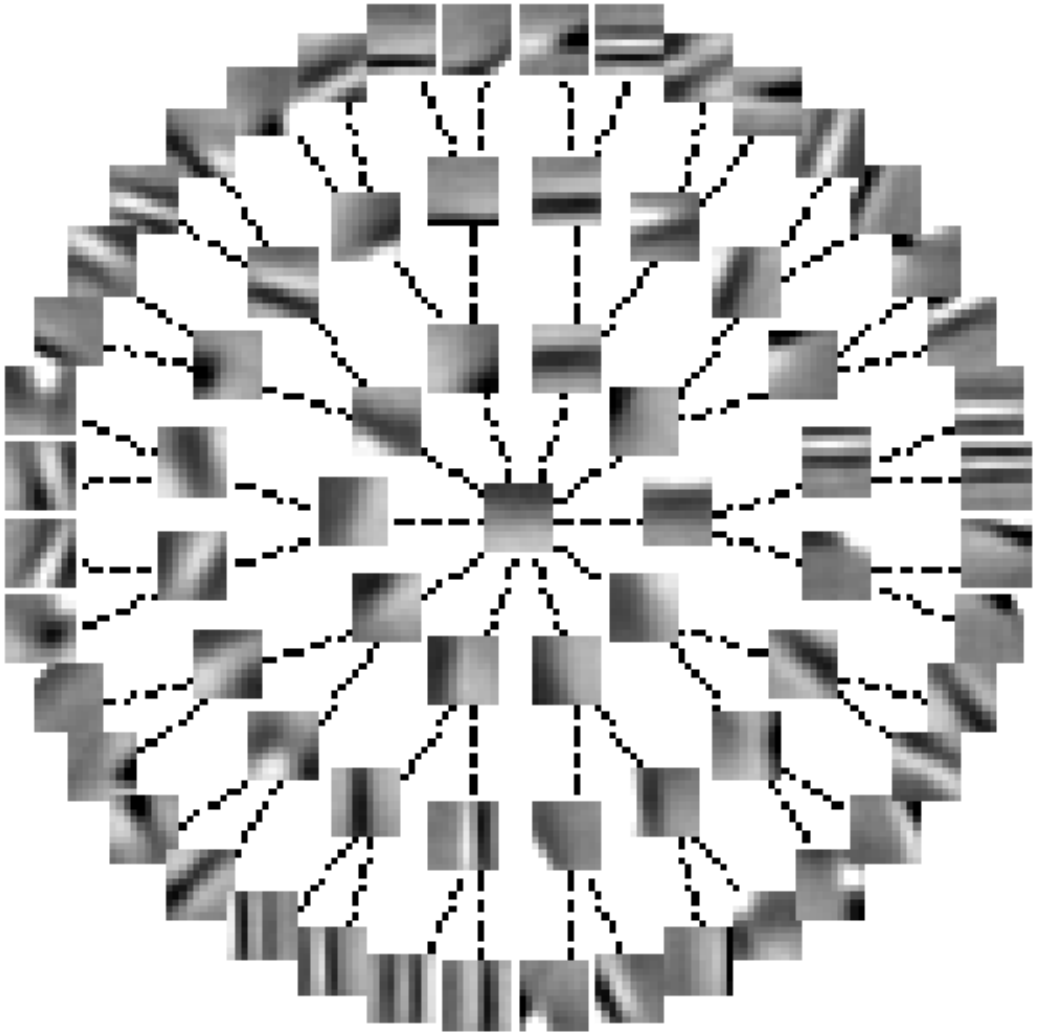
$$\min_{W,Z} \sum_{x \in X} ||Wz - x||^2 + |z|^T S |z|$$



[Gregor, Szlam, LeCun NIPS 2011]

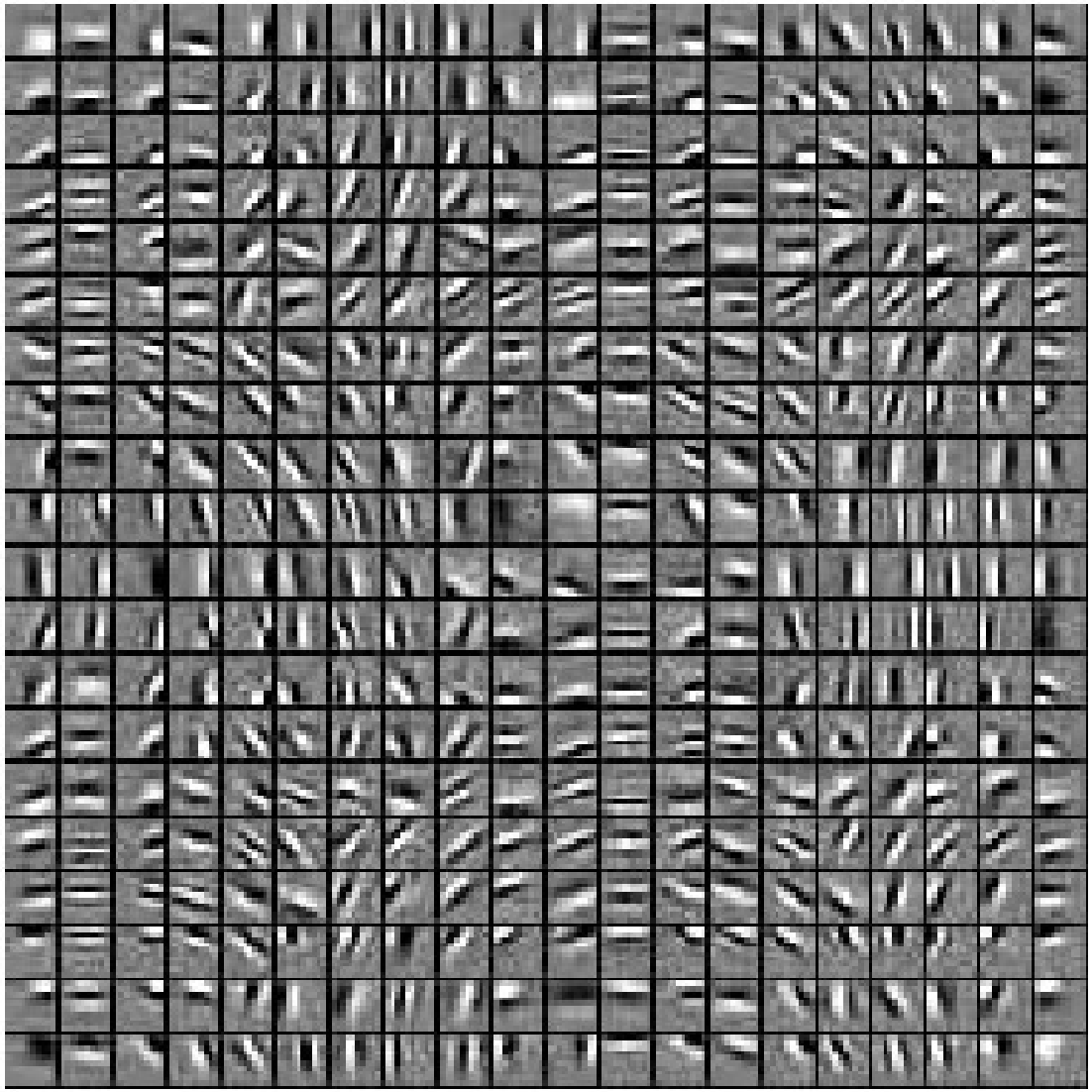
Invariant Features via Lateral Inhibition: Structured Sparsity

- Each edge in the tree indicates a zero in the S matrix (no mutual inhibition)
- Sij is larger if two neurons are far away in the tree



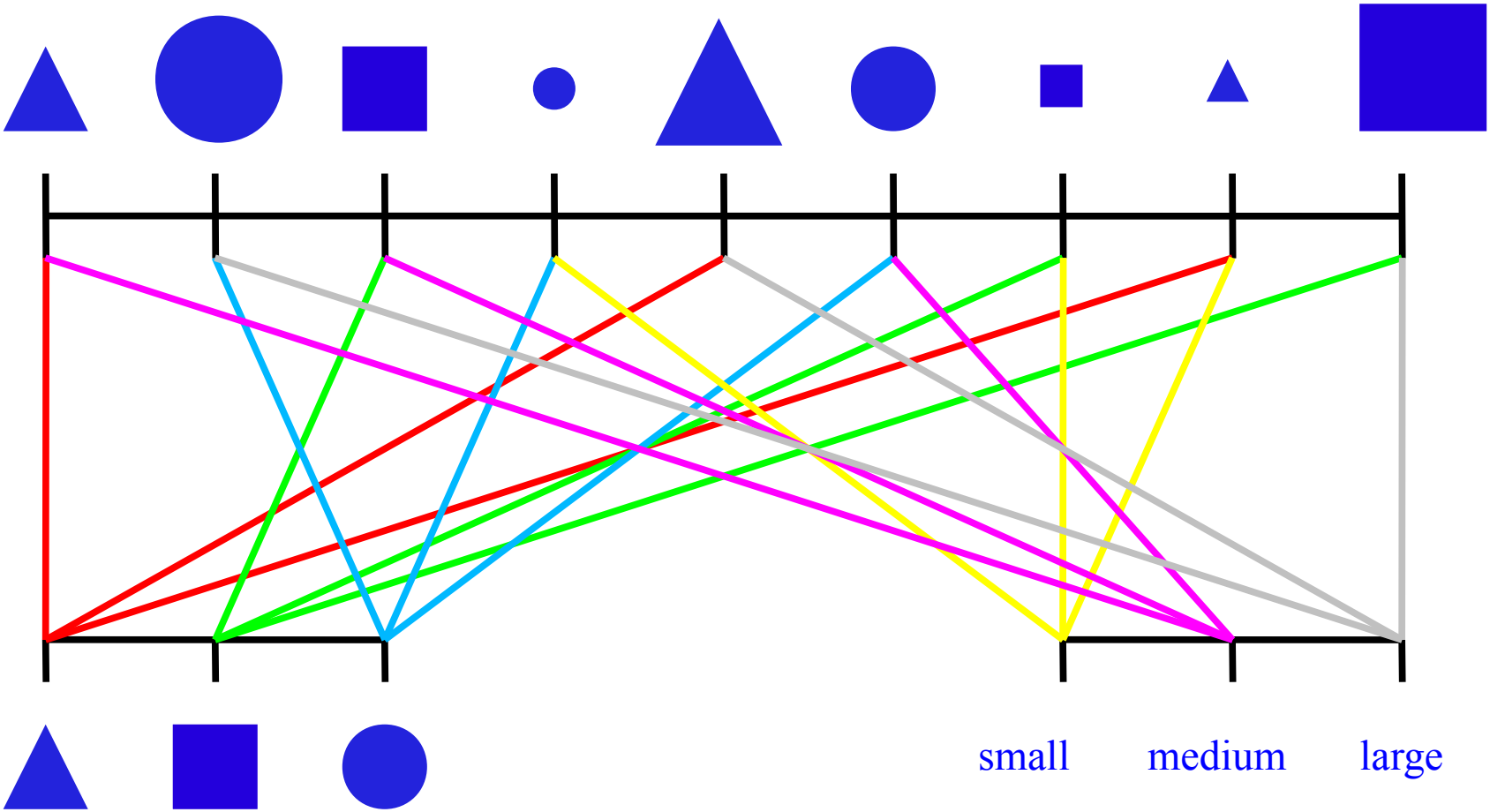
Invariant Features via Lateral Inhibition: Topographic Maps

- Non-zero values in S form a ring in a 2D topology
  - Input patches are high-pass filtered



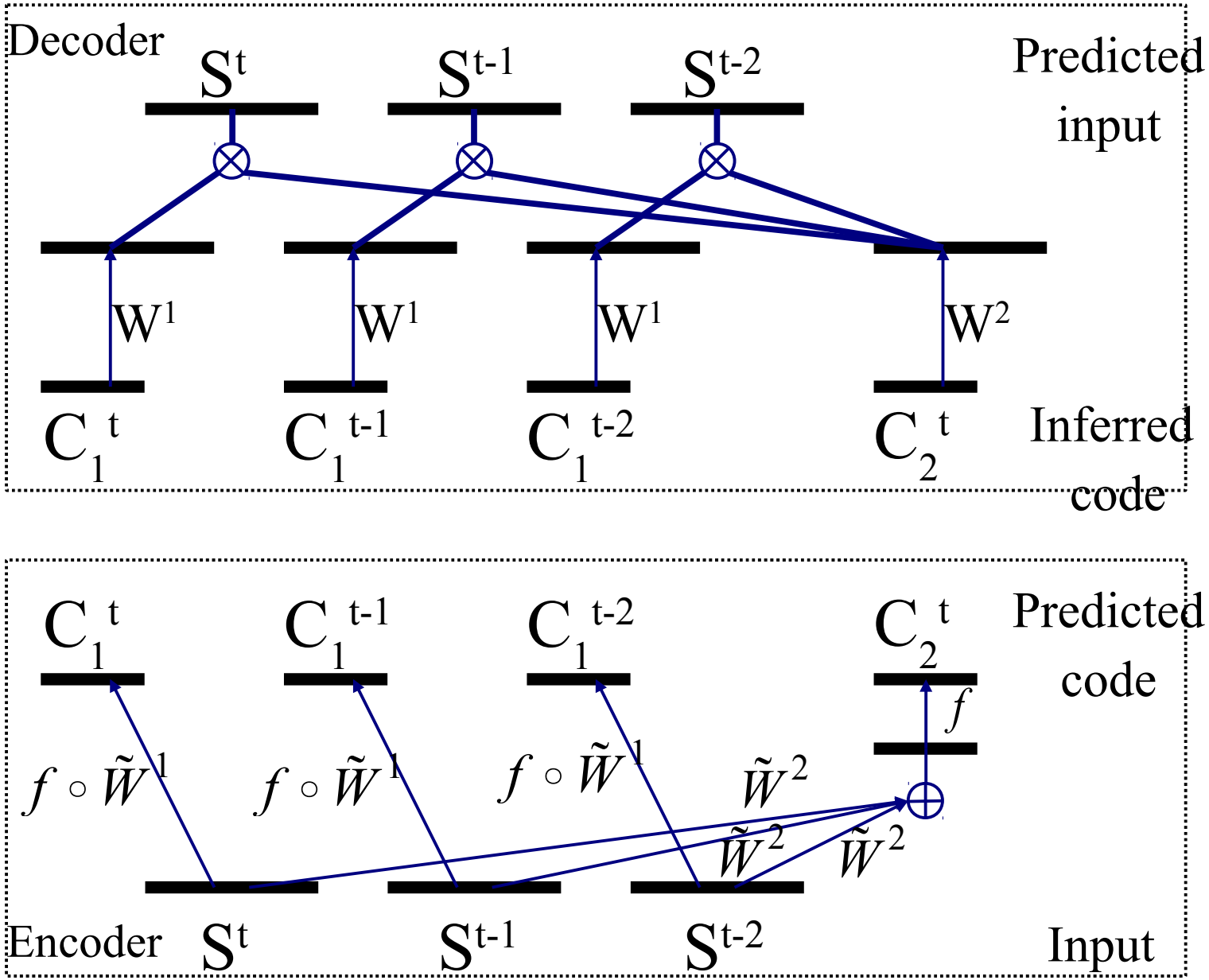
Invariant Features through Temporal Constancy

- Object is cross-product of object type and instantiation parameters
  - Mapping units [Hinton 1981], capsules [Hinton 2011]



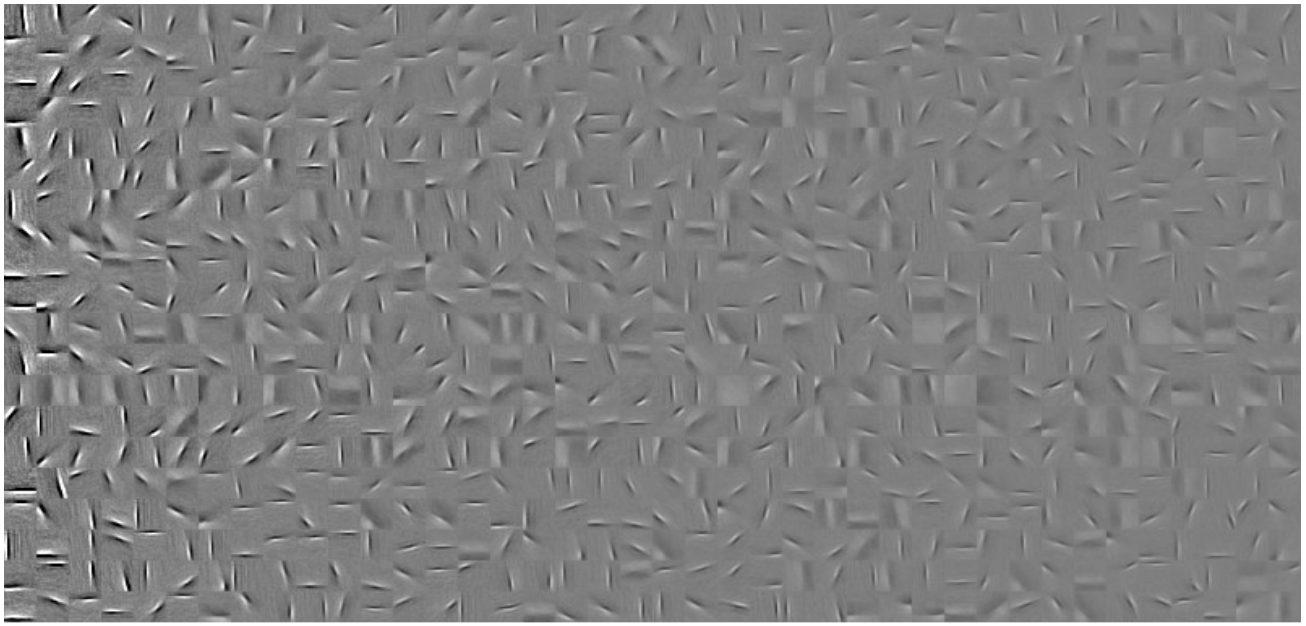
[Karol Gregor et al.]

What-Where Auto-Encoder Architecture

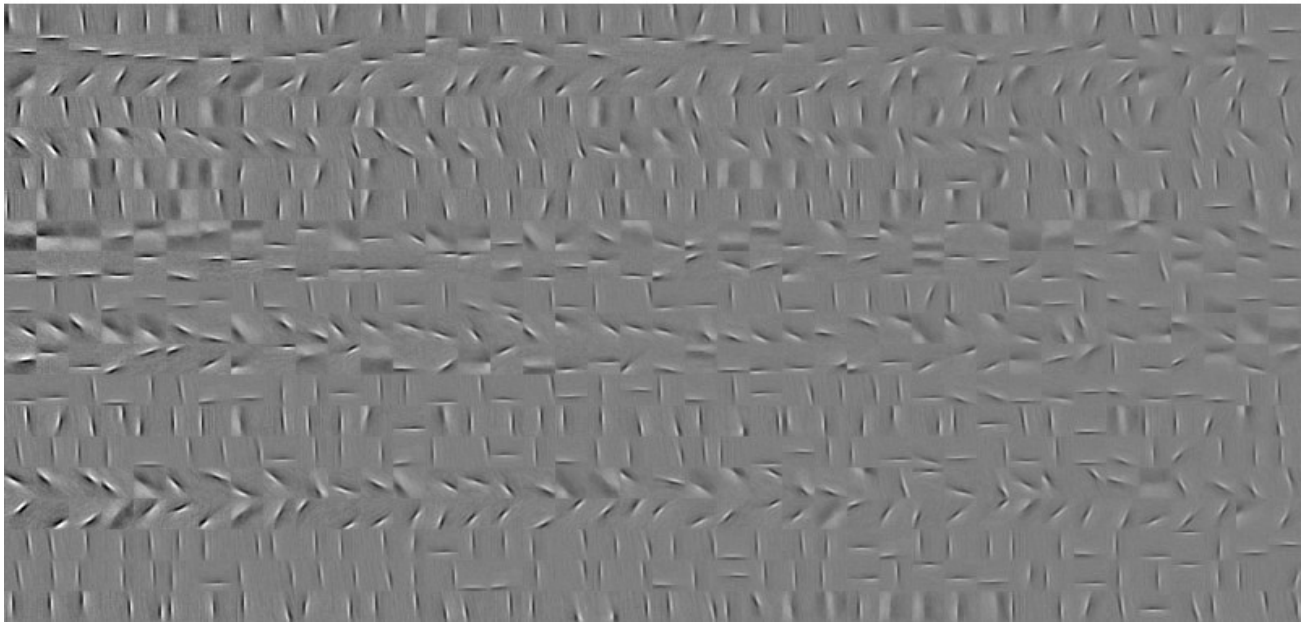


Low-Level Filters Connected to Each Complex Cell

C1
(where)

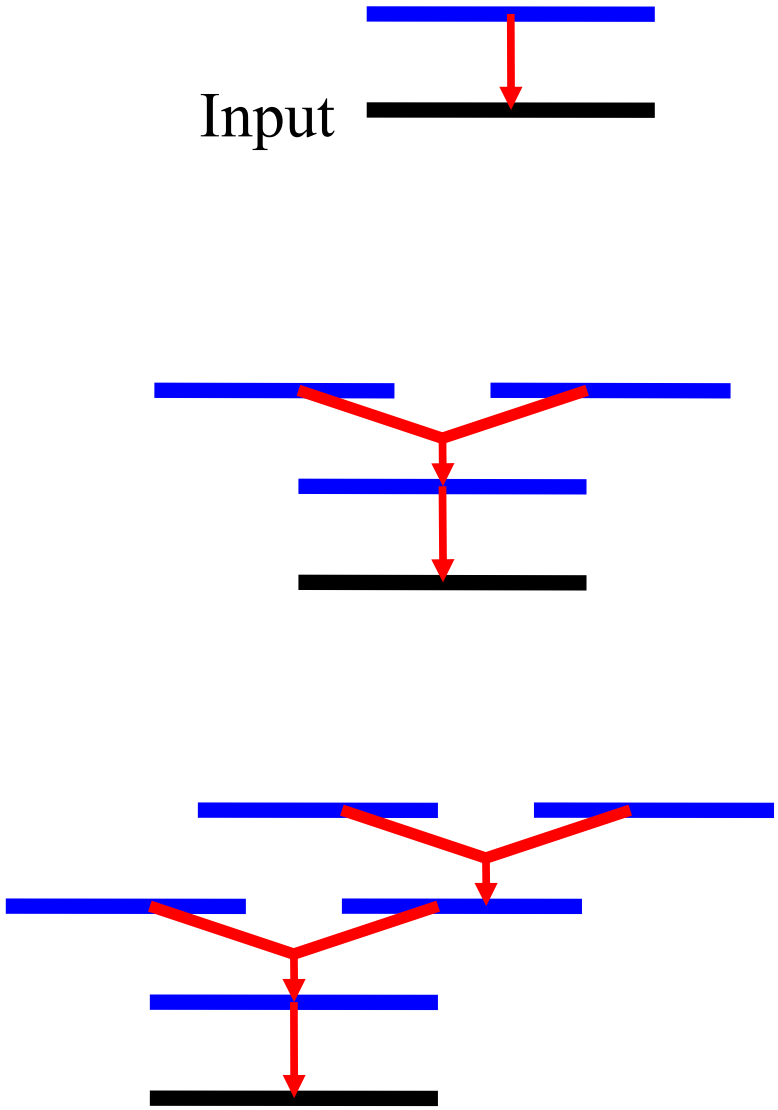


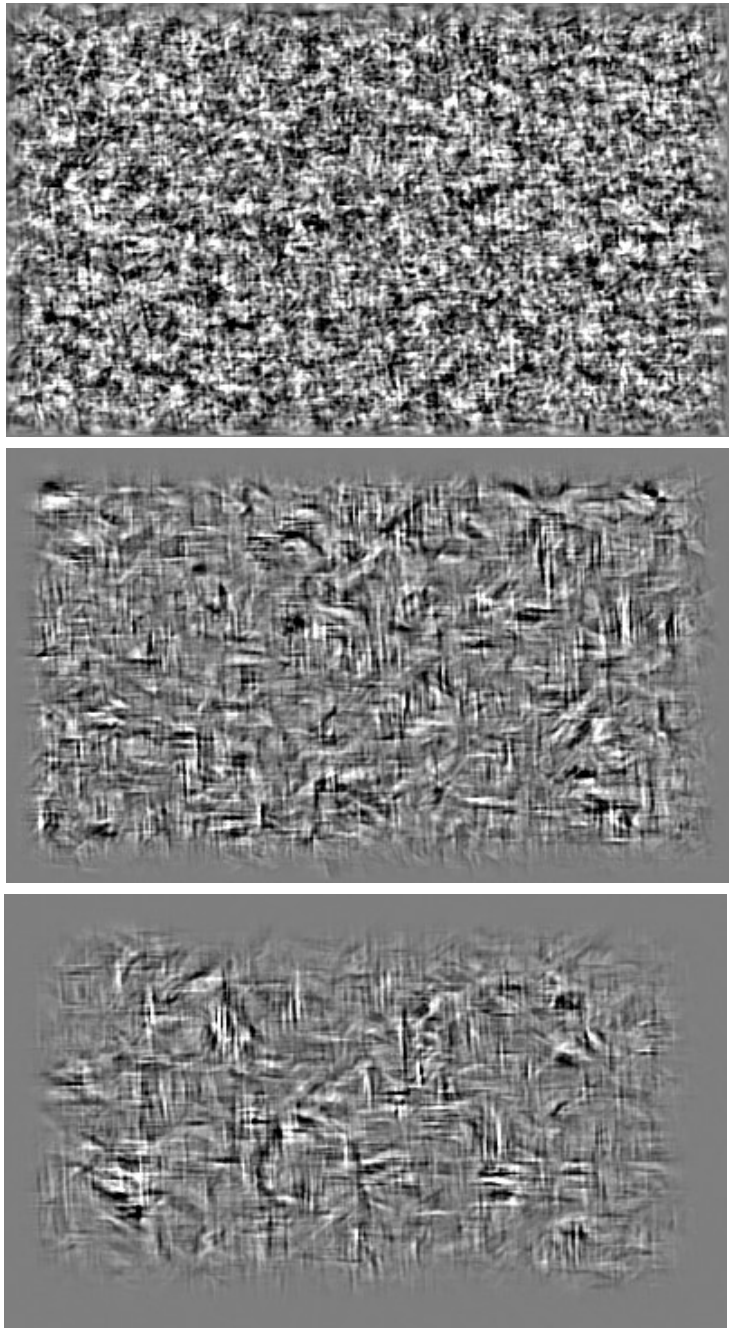
C2
(what)



Generating Images

Generating images



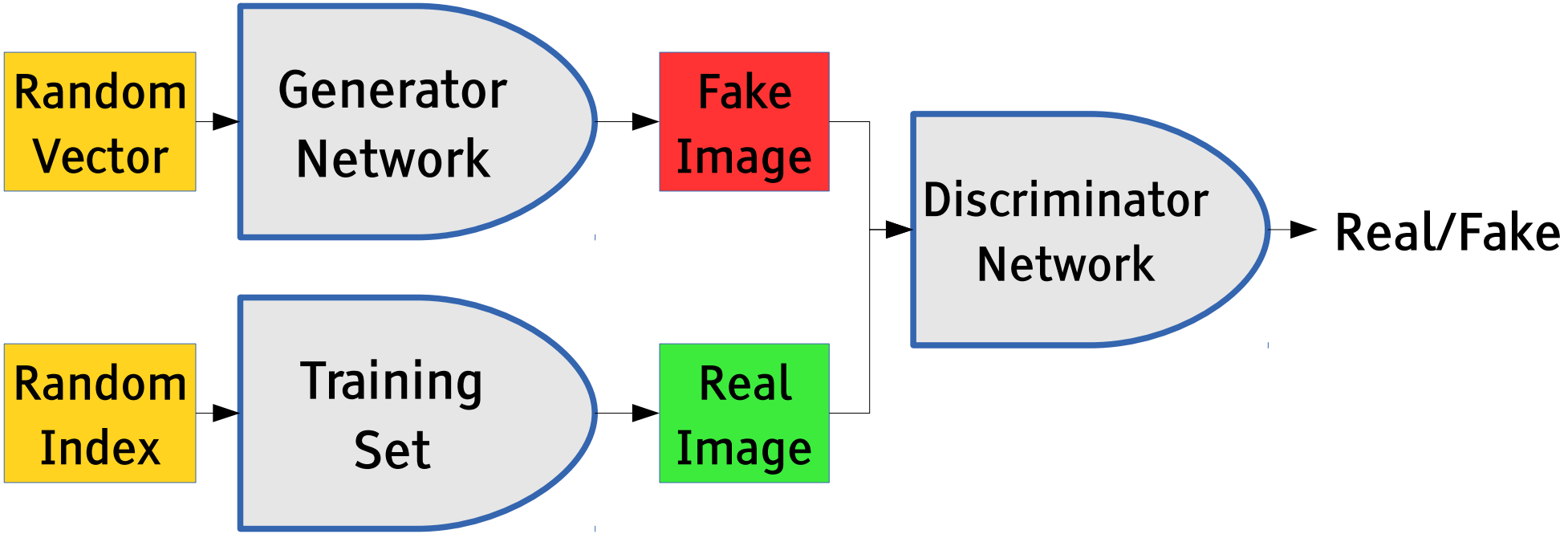


Laplacian Generative Adverserial Network (LAPGAN)

Also known as EyeScream

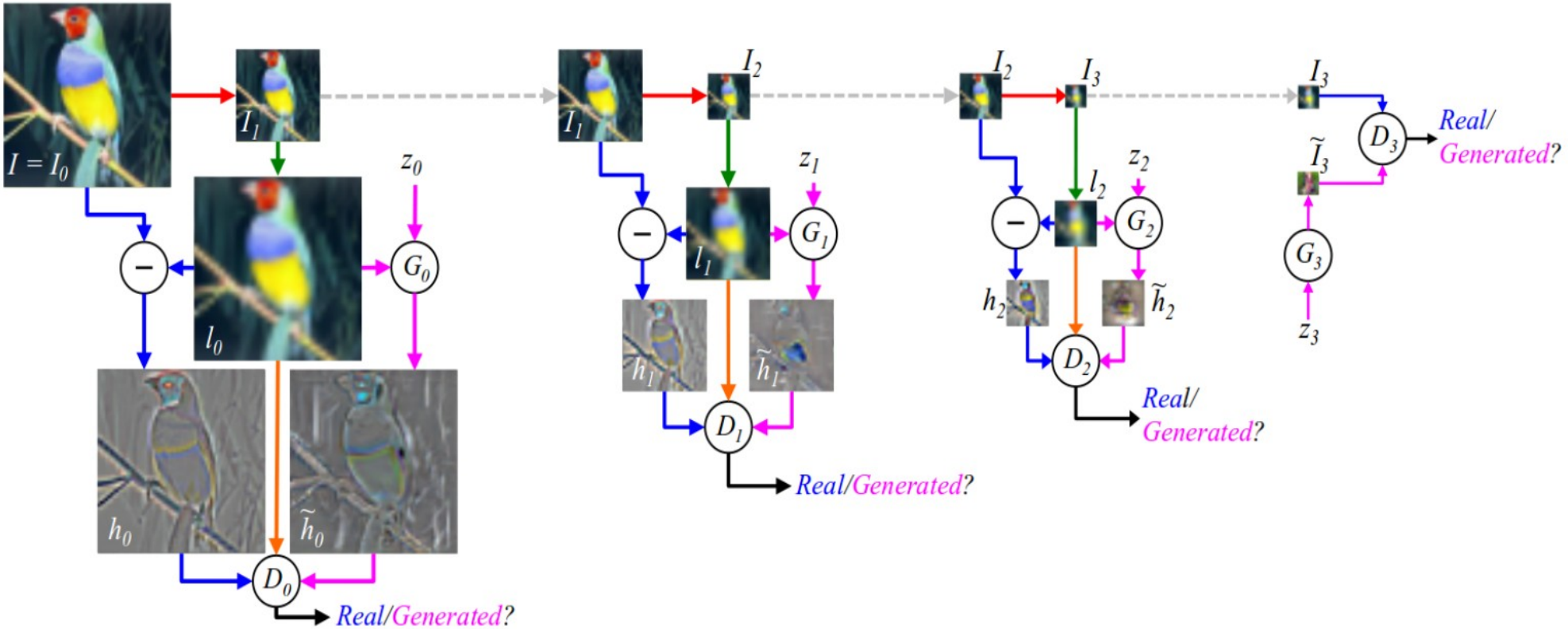
Generative Adversarial Networks

- [Goodfellow et al. NIPS 2014]
- Generator net maps random numbers to image
- Discriminator learns to tell real from fake images.
- Generator can cheat: it knows the gradient of the output of the discriminator with respect to its input



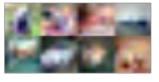
Laplacian GAN: LAPGAN (aka EyeScream)

- Learns to generate images [Denton et al. NIPS 2015]
- Generator net produces coefficients of a Laplacian Pyramid representation of the image
- Discriminator learns to tell real from fake Laplacian images.

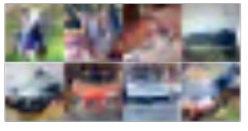


“EyeScream”

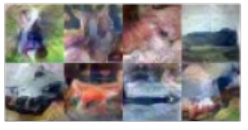
- **http://soumith.ch/eyescream/**



CIFAR-8

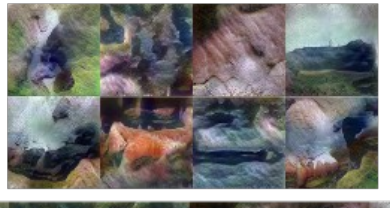


CIFAR-16



Imagenet-32

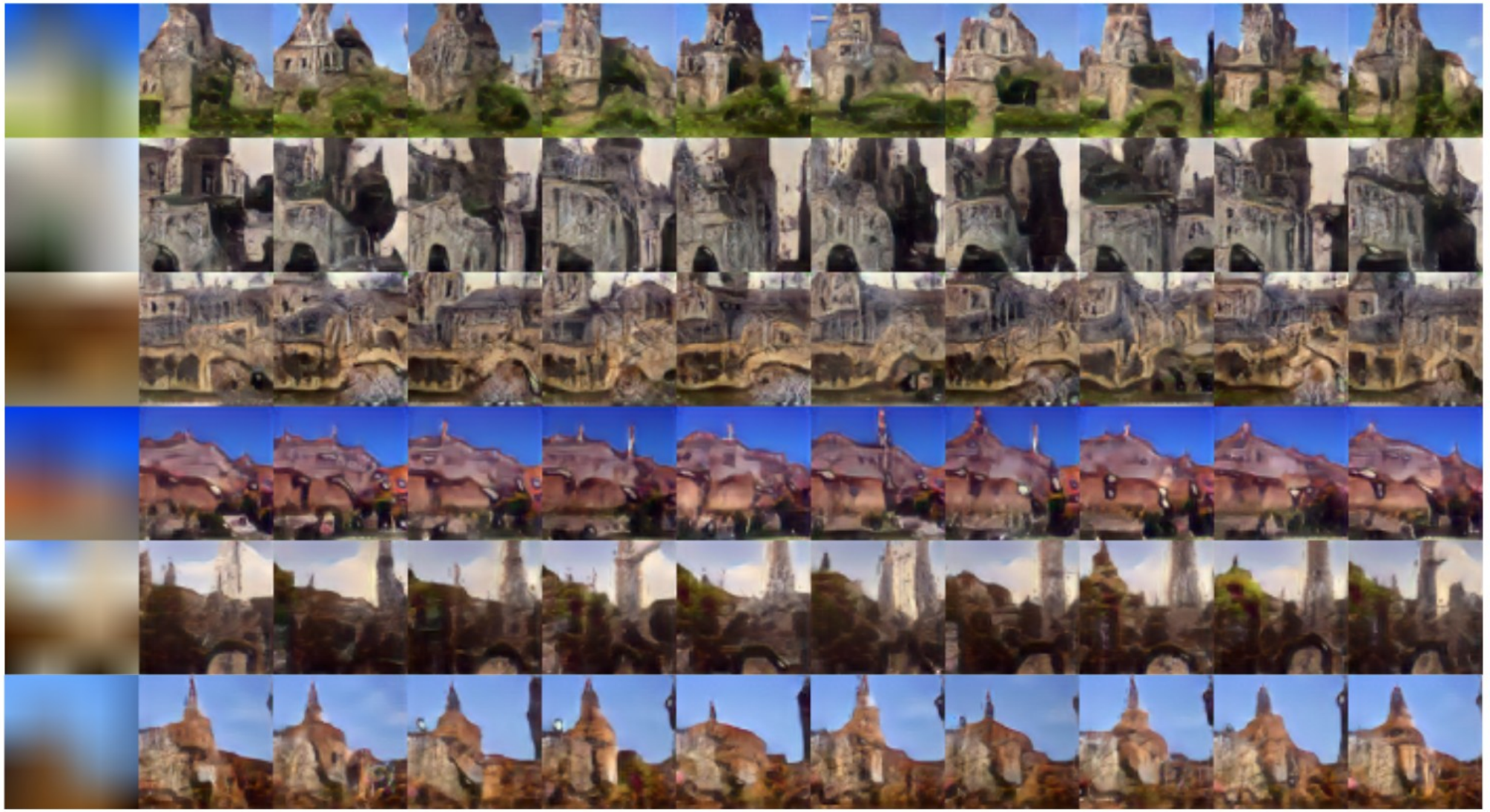
Imagenet-32 (recursive)



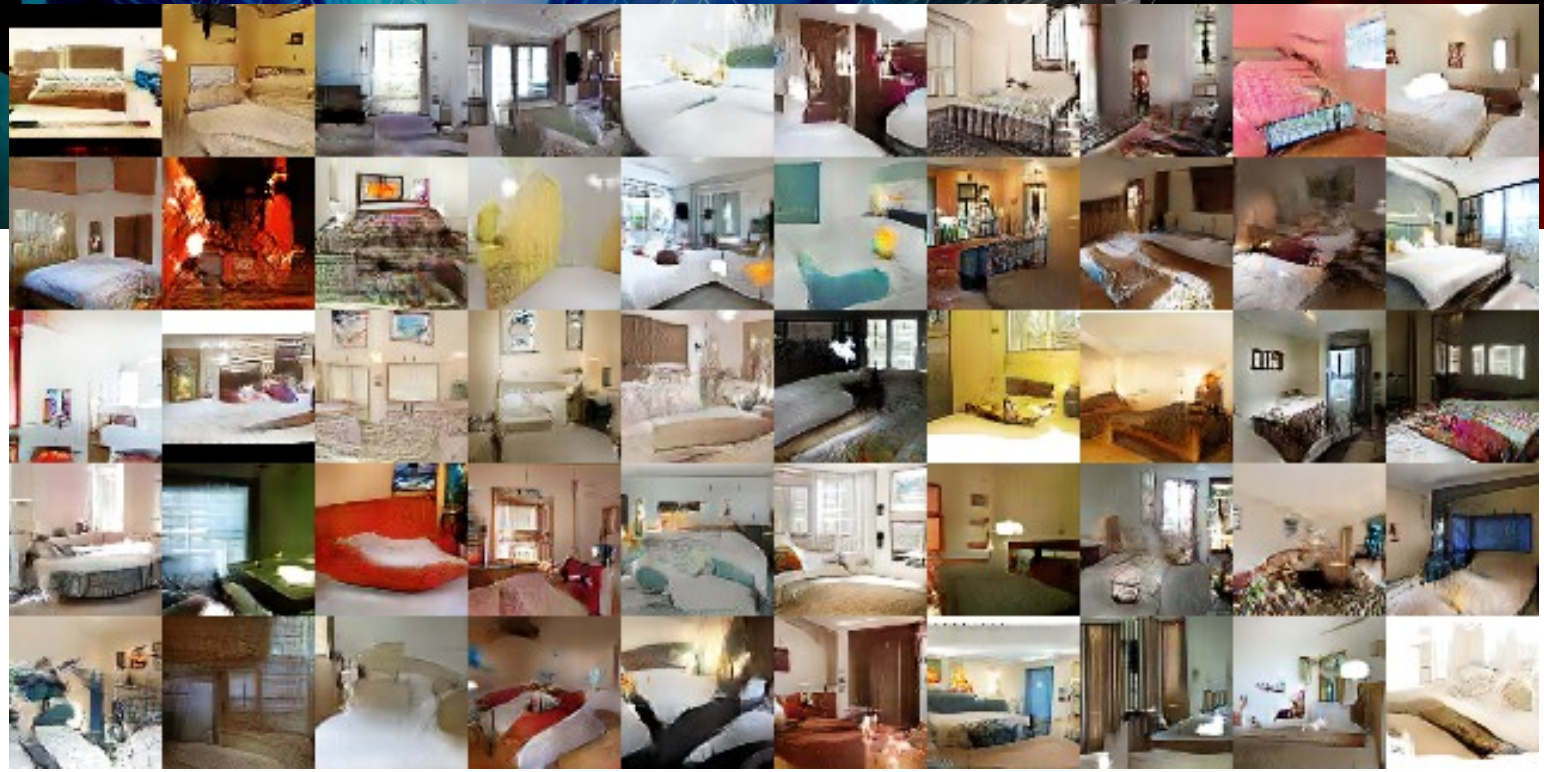
Imagenet-32 (recursive)

“EyeScream” / “LAPGAN”

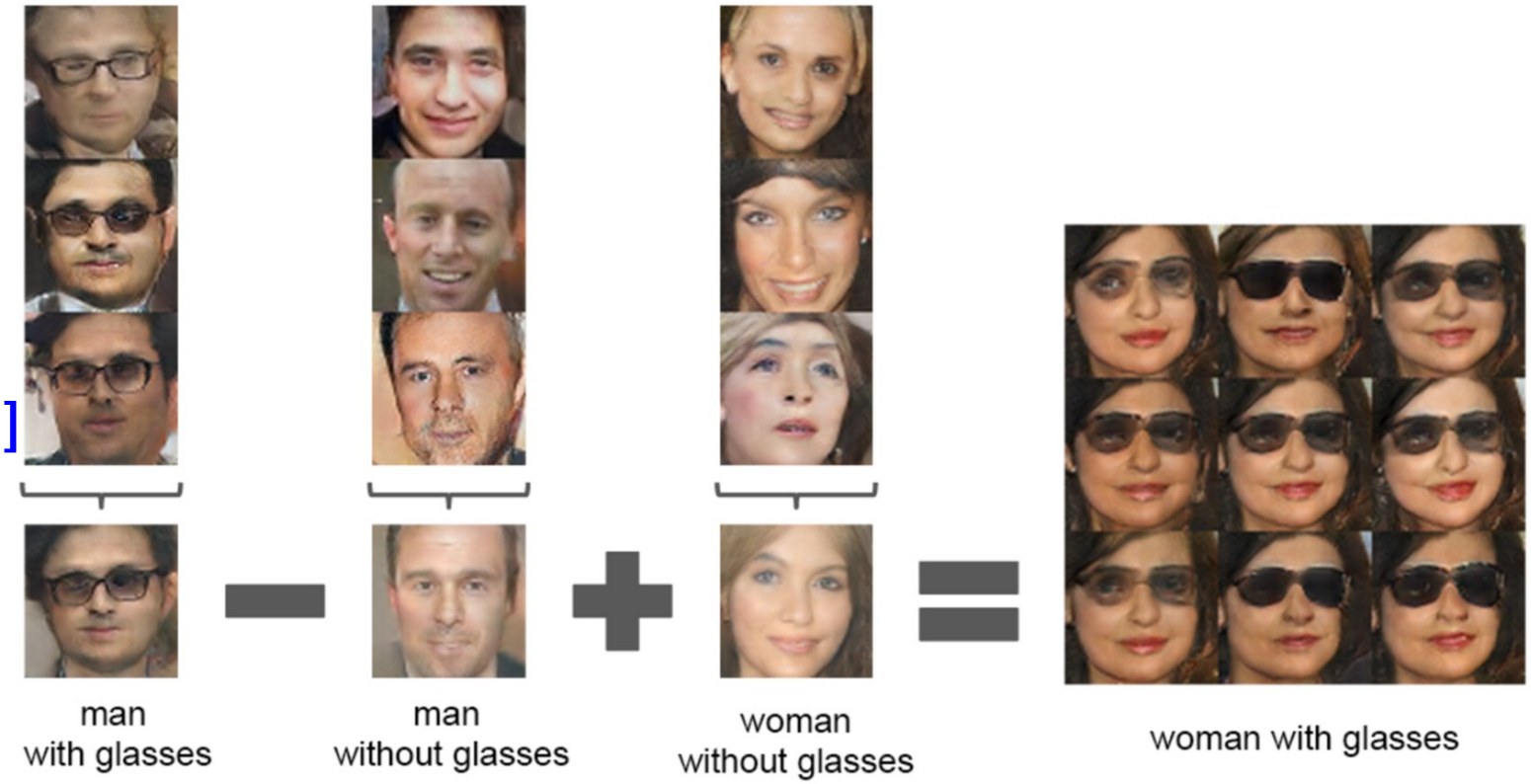
- http://soumith.ch/eyescream/



Discovering Regularities



DCGAN:
adversarial training
to generate images.
[Metz, Chintala 2015]



Video Prediction

[Mathieu, Couprie, LeCun arXiv:1511:05440]

Unsupervised Learning is the "Dark Matter" of AI

Y LeCun

Unsupervised learning is the only form of learning that can provide enough information to train large neural nets with billions of parameters.

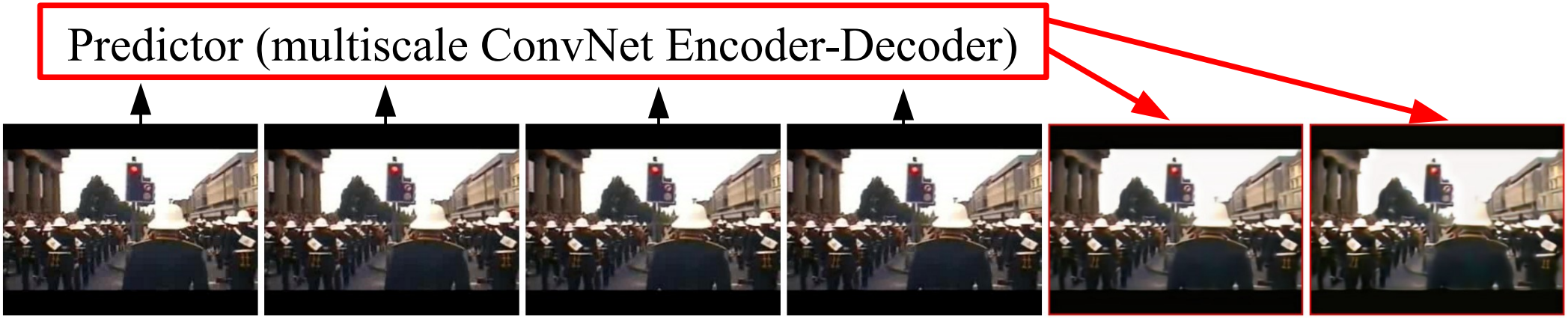
- Supervised learning would take too much labeling effort
- Reinforcement learning would take too many trials

But we don't know how to do unsupervised learning (or even formulate it)

- We have lots of ideas and methods
- They just don't work that well yet.

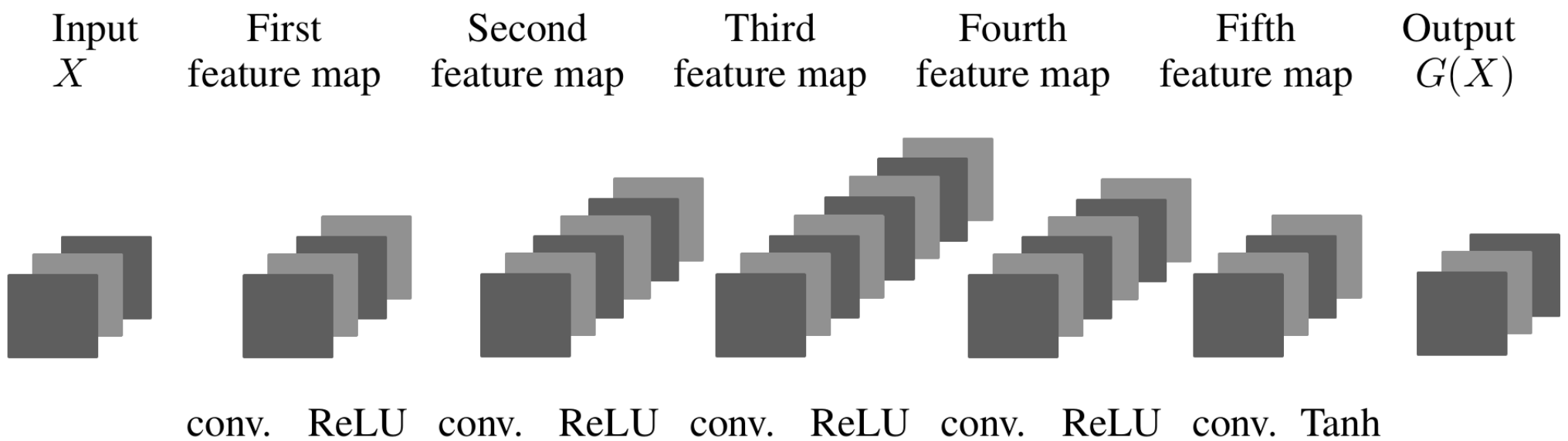
Why is it so hard? The world is unpredictable!

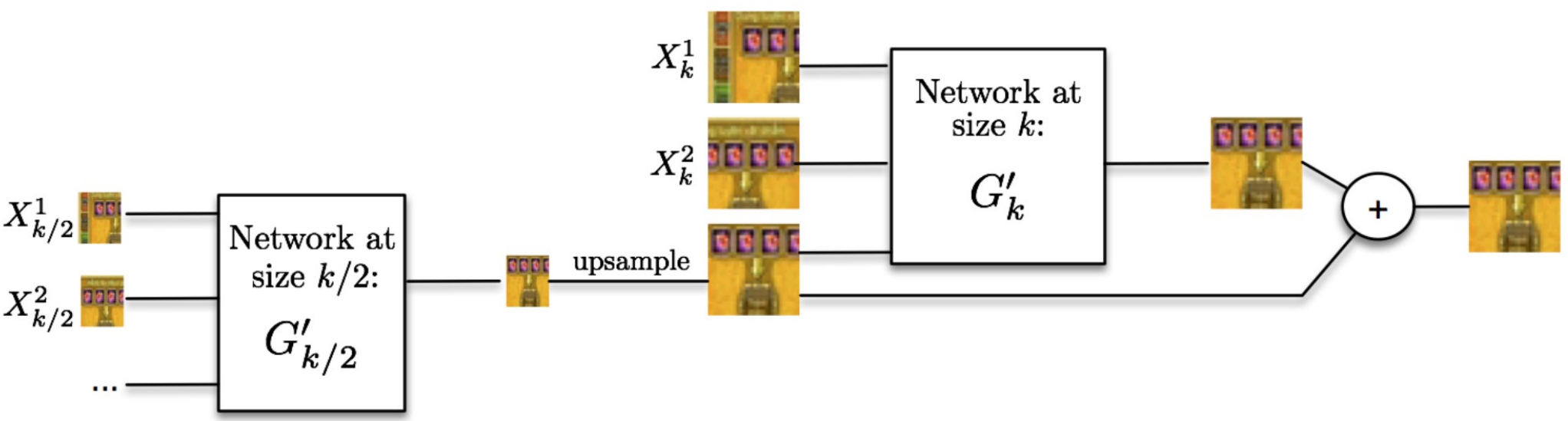
- Predictors produce an average of all possible futures → **Blurry image.**



Multi-Scale ConvNet for Video Prediction

- **4 to 8 frames input → ConvNet with no pooling → 1 to 8 frames output**

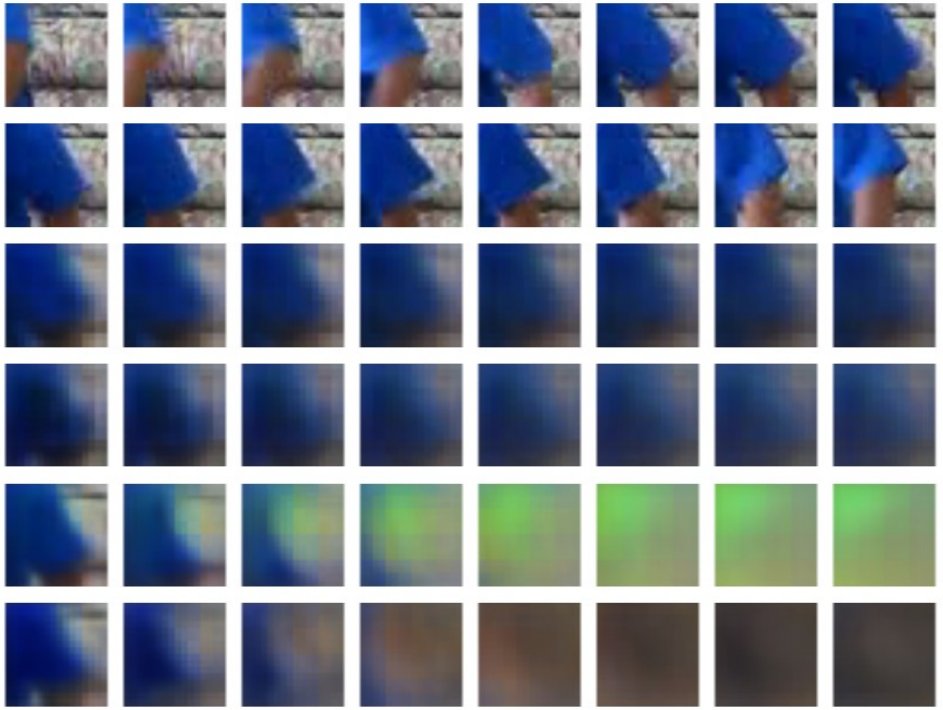




Can't Use Squared Error: blurry predictions

- The world is unpredictable
- MSE training predicts the average of possible futures: blurry images.

Ground truth ℓ_2 result



Input

Ground truth

ℓ_1

ℓ_2

ℓ_1 recursive

ℓ_2 recursive

Multi-Scale ConvNet for Video Prediction

Architectures

Models 4 frames in input – 1 frame in output

Generative network scales	G_1	G_2	G_3	G_4
Number of feature maps	128, 256, 128	128, 256, 128	128, 256, 512, 256, 128	128, 256, 512, 256, 128
Conv. kernel size	3, 3, 3, 3	5, 3, 3, 5	5, 3, 3, 3, 5	7, 5, 5, 5, 5, 7
Adversarial network scales	D_1	D_2	D_3	D_4
Number of feature maps	64	64, 128, 128	128, 256, 256	128, 256, 512, 128
Conv. kernel size (no padding)	3	3, 3, 3	5, 5, 5	7, 7, 5, 5
Fully connected	512, 256	1024, 512	1024, 512	1024, 512

Models 8 frames in input – 8 frames in output

Generative network scales	G_1	G_2	G_3	G_4
Number of feature maps	16, 32, 64	16, 32, 64	32, 64, 128	32, 64, 128, 128
Conv. kernel size	3, 3, 3, 3	5, 3, 3, 3	5, 5, 5, 5	7, 5, 5, 5, 5
Adversarial network scales	D_1	D_2	D_3	D_4
Number of feature maps	16	16, 32, 32	32, 64, 64	32, 64, 128, 128
Conv. kernel size (no padding)	3	3, 3, 3	5, 5, 5	7, 7, 5, 5
Fully connected	128, 64	256, 128	256, 128	256, 128

Multi-Scale ConvNet for Video Prediction

Results on UCF101 (10% of test images)

- 8 frames input → 8 frames output

	1st frame prediction scores		8th frame prediction scores	
	PSNR	Sharpness	PSNR	Sharpness
ℓ_2	18.3	0.47	16.4	0.36
Adv	21.0	0.65	18.9	0.54
ℓ_1	21.2	0.57	18.3	0.51
GDL ℓ_1	**21.8**	**0.87**	**19.2**	**0.79**

Results on UCF101 (10% of test images)

- 4 frames input → 1 frames output

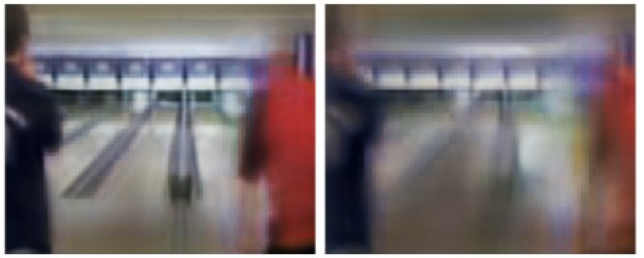
	1st frame prediction scores		2nd frame prediction scores	
	PSNR	Sharpness	PSNR	Sharpness
ℓ_2	20.1	0.48	14.1	0.29
ℓ_1	22.2	0.58	16.0	0.48
GDL ℓ_1	23.9	0.69	18.5	0.60
Adv	24.4	**0.95**	18.9	**1.00**
Adv+GDL	**27.2**	0.77	**22.6**	0.68

Multi-Scale ConvNet for Video Prediction

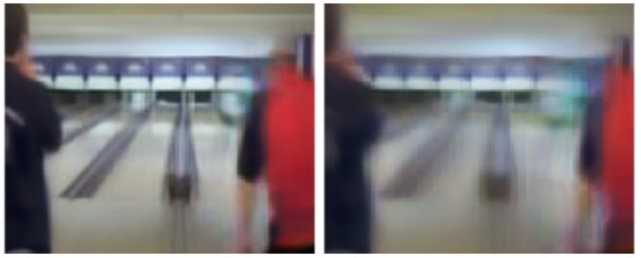
Examples

Input frames

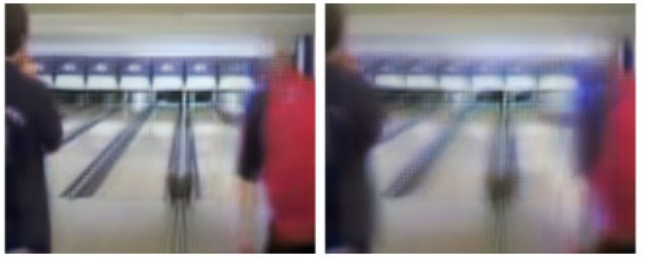
Ground truth



ℓ_2 result



ℓ_1 result



GDL ℓ_1 result

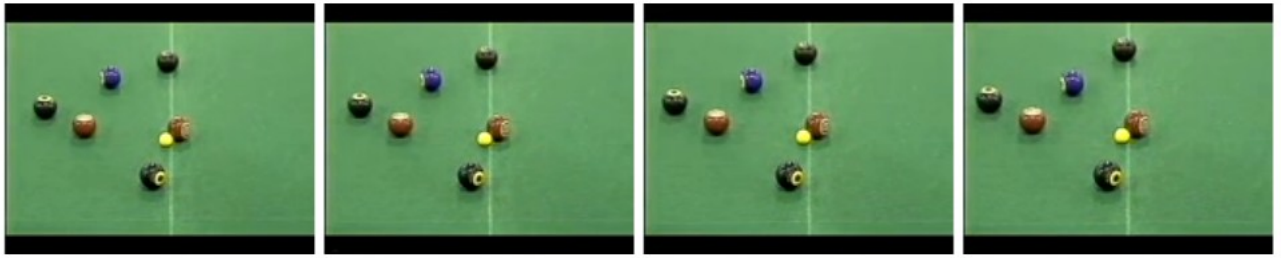
Adversarial result

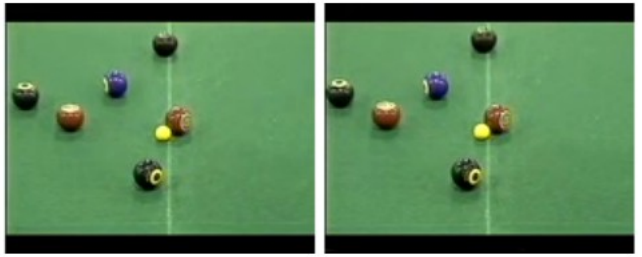
Adversarial+GDL result

Multi-Scale ConvNet for Video Prediction

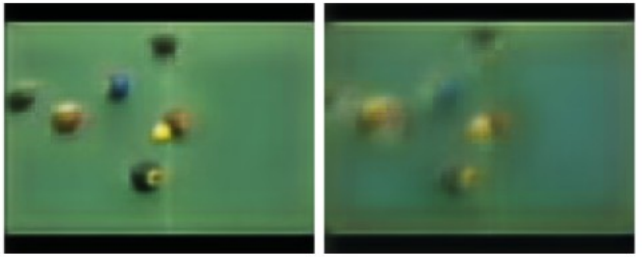
Examples

Input frames

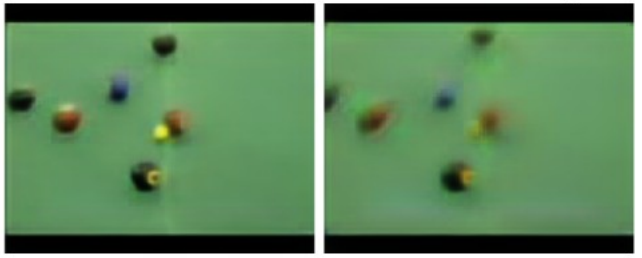




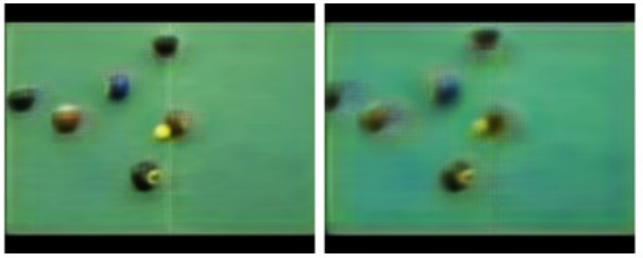
Ground truth



ℓ_2 result

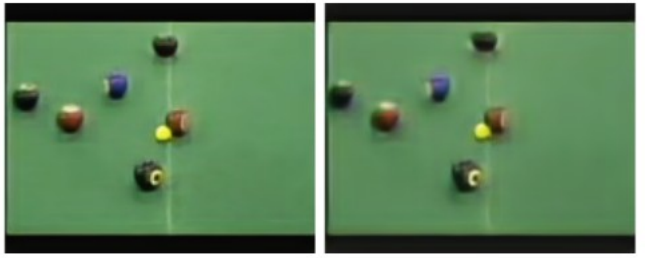


ℓ_1 result



GDL ℓ_1 result

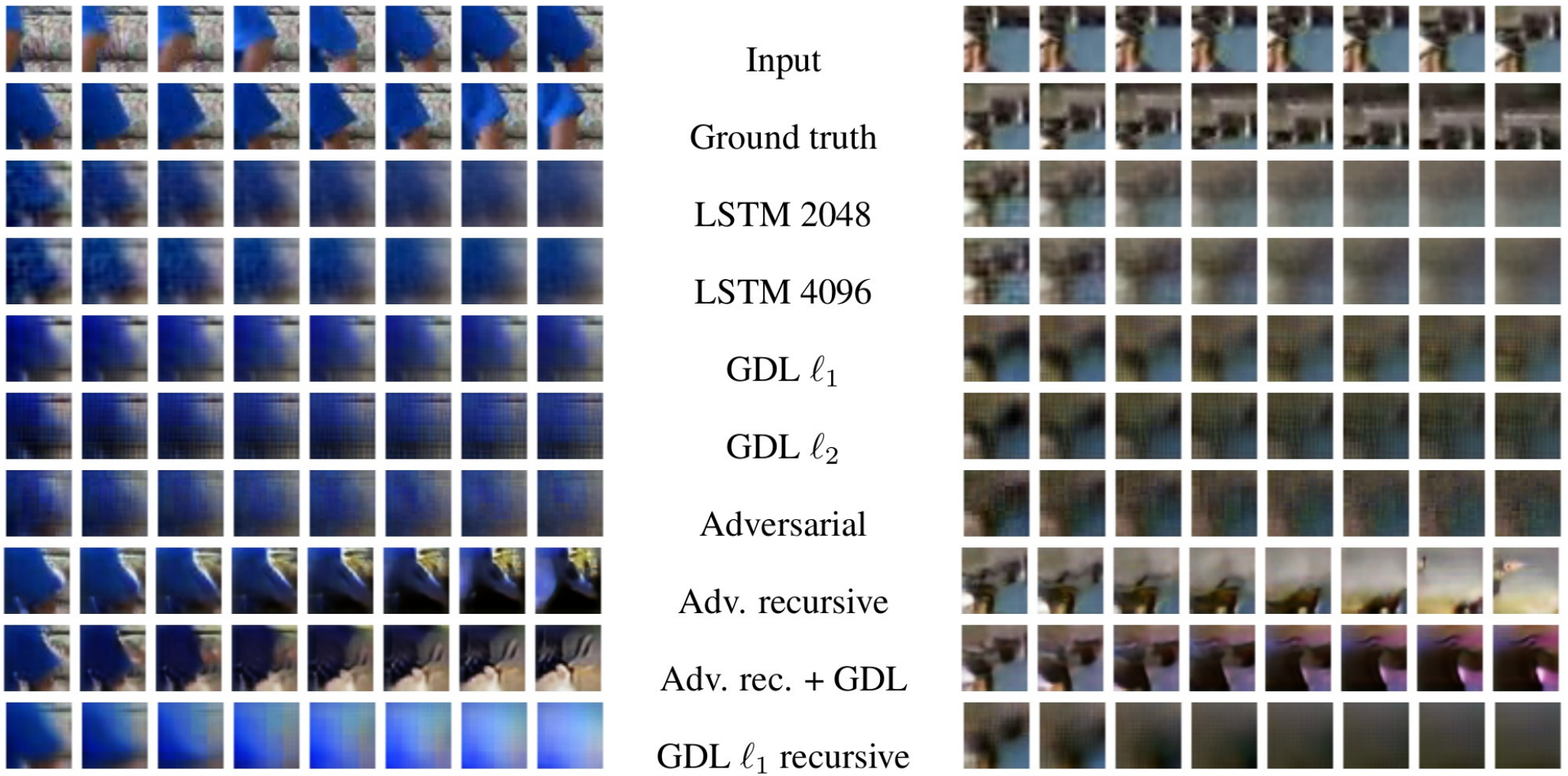
Adversarial result



Adversarial+GDL result

Multi-Scale ConvNet for Video Prediction

- **Comparison with [Srivastava et al. 2015] who used LSTM.**



Predictive Unsupervised Learning

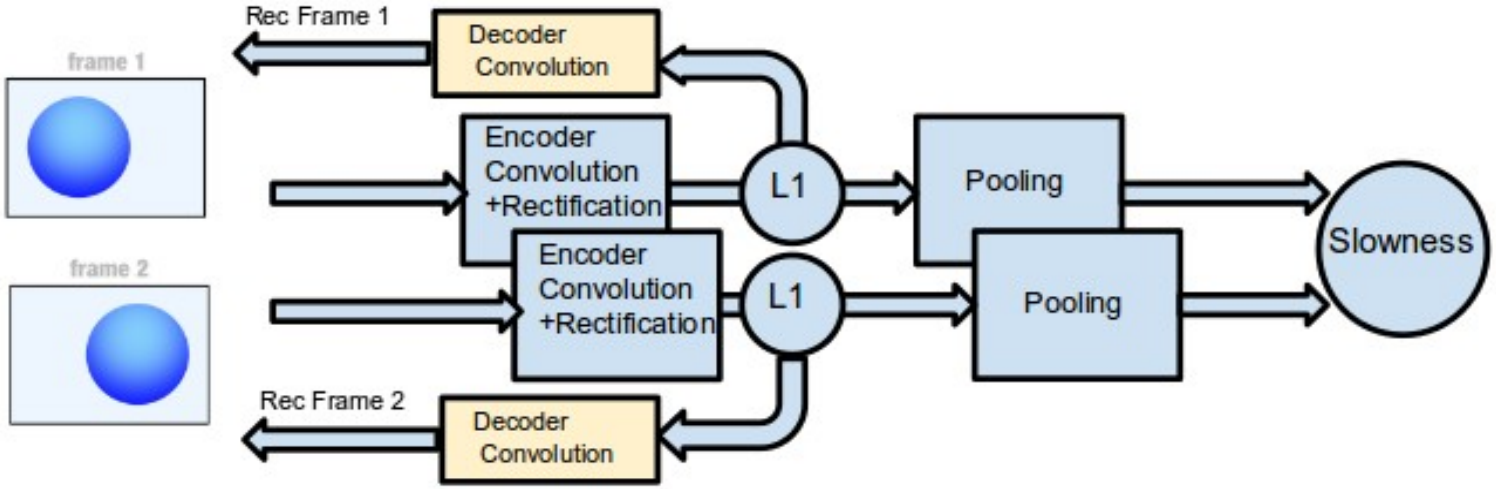
Some success with "adversarial training"

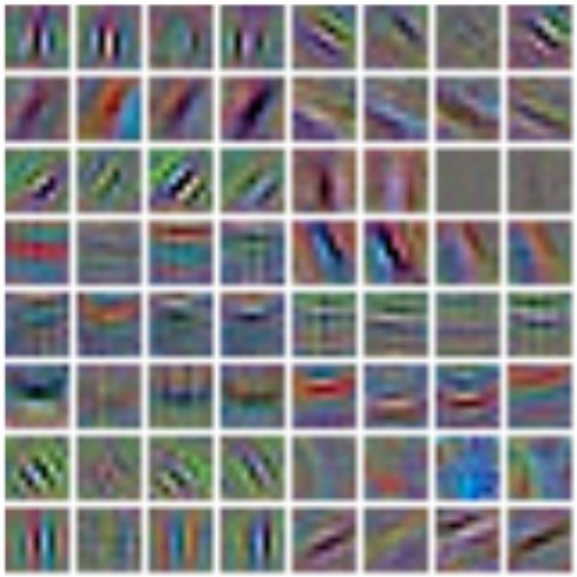
- [Mathieu, Couprie, LeCun arXiv:1511:05440]

But we are far from a complete solution.

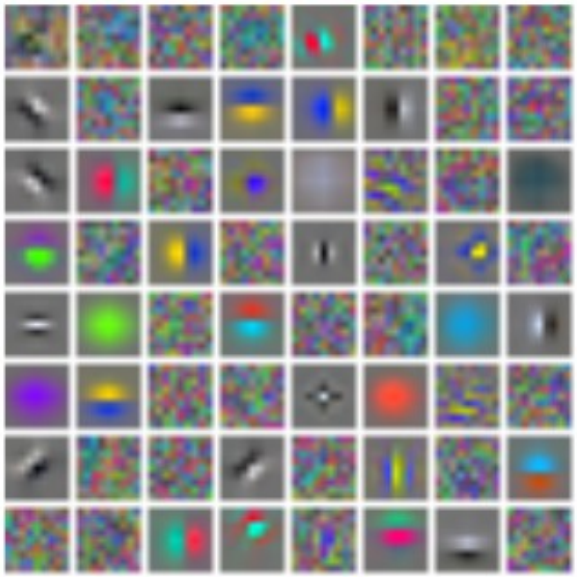
Exploiting Temporal Consistency

Sparse Auto-Encoder with "Slow Feature" Penalty

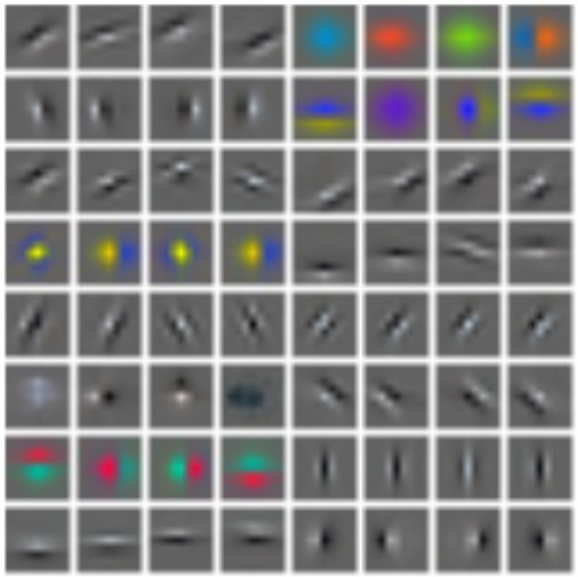




Supervised filters CIFAR10



sparse conv. auto-encoder



slow & sparse conv. auto-encoder

Trained on YouTube videos

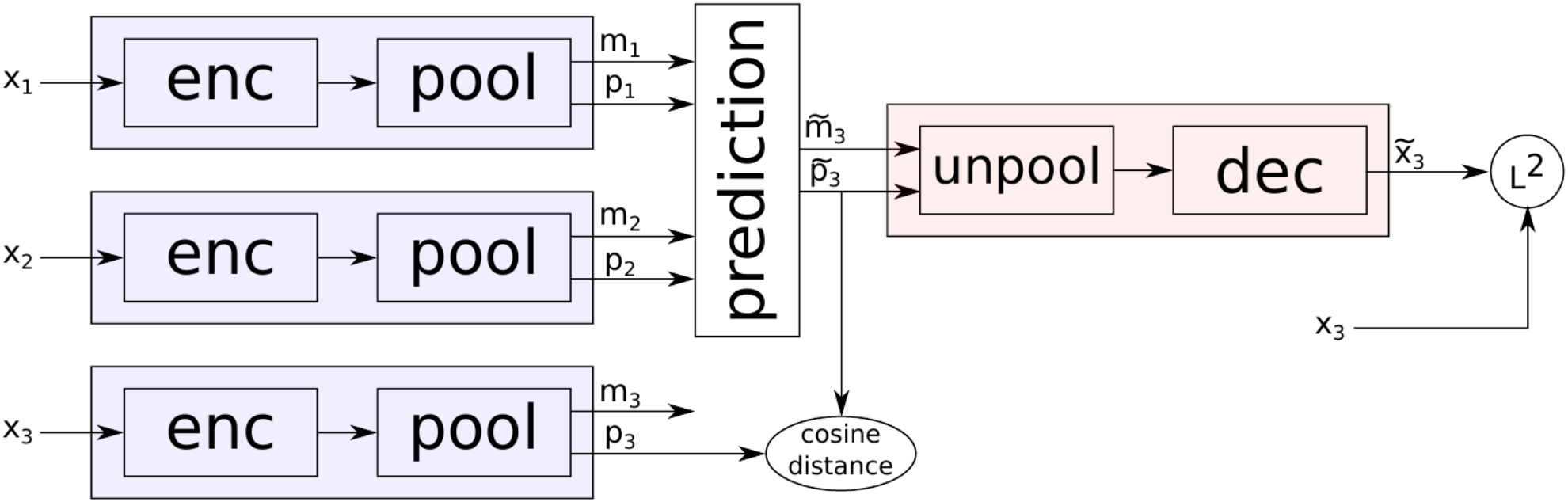
[Goroshin et al. ICCV 2015, Arxiv:1412.6056]

Unsupervised Learning by Prediction and Linearization

[Goroshin, Mathieu, LeCun NIPS 2015, arXiv:1506.03011]

- Trained on 3 successive frames of video
- Maximize the colinearity between successive changes of feature vectors
- So that "natural" changes stay in linear manifold in feature space.

$$L = \frac{1}{2}\|G_W(\mathbf{a}\begin{bmatrix} z^t & z^{t-1}\end{bmatrix}^T) - x^{t+1}\|_2^2 - \lambda \frac{(z^t - z^{t-1})^T(z^{t+1} - z^t)}{\|z^t - z^{t-1}\|\|z^{t+1} - z^t\|}$$



Unsupervised Learning by Prediction and Linearization

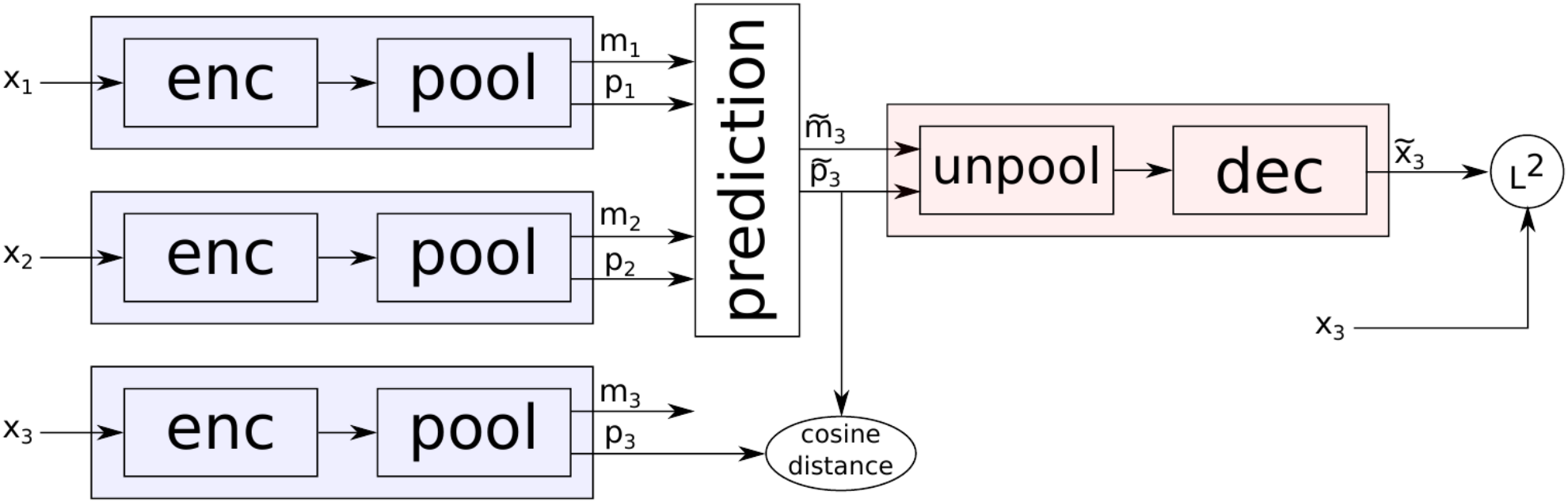
$$L = \frac{1}{2}\|G_W(\mathbf{a}\begin{bmatrix} z^t & z^{t-1}\end{bmatrix}^T) - x^{t+1}\|_2^2 - \lambda \frac{(z^t - z^{t-1})^T(z^{t+1} - z^t)}{\|z^t - z^{t-1}\|\|z^{t+1} - z^t\|}$$

- **Magnitude**
  - (soft max)

$$m_k = \sum_{N_k} z(f,x,y) \frac{e^{\beta z(f,x,y)}}{\sum_{N_k} e^{\beta z(f',x',y')}} \approx \max_{N_k} z(f,x,y)$$

- **Phase**
  - (soft argmax)

$$\mathbf{p}_k = \sum_{N_k} \begin{bmatrix} f \\ x \\ y \end{bmatrix} \frac{e^{\beta z(f,x,y)}}{\sum_{N_k} e^{\beta z(f',x',y')}} \approx \arg\max_{N_k} z(f,x,y)$$



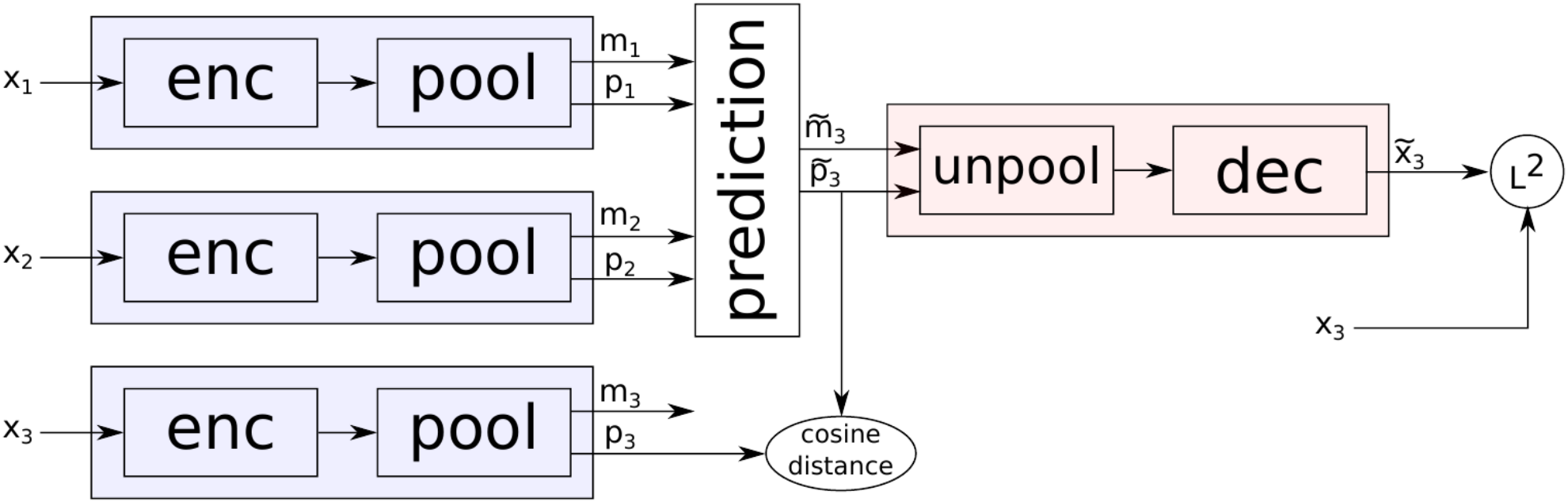
Unsupervised Learning by Prediction and Linearization

[Goroshin, Mathieu, LeCun NIPS 2015, arXiv:1506.03011]

- **Version 2:**
  - Make sure the "what"/magnitude changes slowly
  - Make sure the "where"/phase changes linearly

$$m^{t+1} = \frac{m^t + m^{t-1}}{2}$$

$$\mathbf{p}^{t+1} = 2\mathbf{p}^t - \mathbf{p}^{t-1}$$



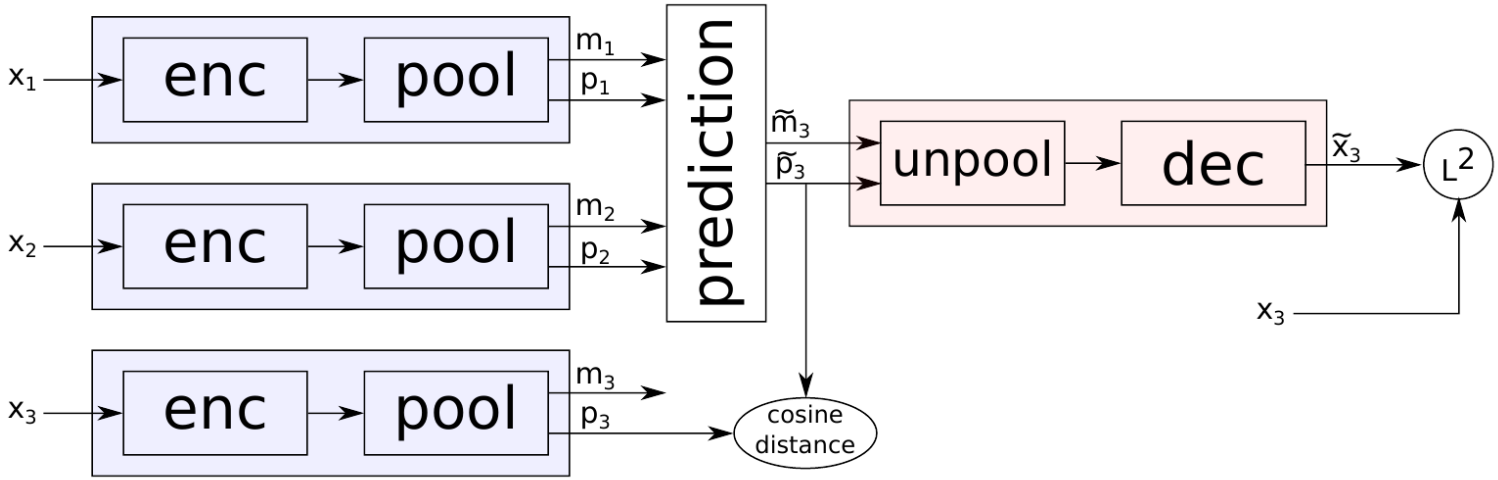
Unsupervised Learning by Prediction and Linearization

[Goroshin, Mathieu, LeCun arXiv:1506.03011]

- **Version 3: the world is unpredictable**
  - Add latent variable to compensate for that.
  - Minimize over them during training

$$\hat{z}_\delta^{t+1} = z^t + (W_1\delta) \odot \mathbf{a} \begin{bmatrix} z^t & z^{t-1} \end{bmatrix}^T$$

$$L = \min_\delta \|G_W(\hat{z}_\delta^{t+1}) - x^{t+1}\|_2^2 - \lambda \frac{(z^t - z^{t-1})^T (z^{t+1} - z^t)}{\|z^t - z^{t-1}\| \|z^{t+1} - z^t\|}$$



Algorithm (with latent variables)

Algorithm 1 Minibatch stochastic gradient descent training for prediction with uncertainty. The number of δ-gradient descent steps (k) is treated as a hyper-parameter.

for number of training epochs **do**
  Sample a mini-batch of temporal triplets $\{x^{t-1}, x^t, x^{t+1}\}$
  Set $\delta_0 = 0$
  Forward propagate x^{t-1}, x^t through the network and obtain the codes z^{t-1}, z^t and the prediction $\hat{x}_0^{t+1}$
  for i =1 to k **do**
    Compute the L^2 prediction error
    Back propagate the error through the decoder to compute the gradient $\frac{\partial L}{\partial \delta^{i-1}}$
    Update $\delta_i = \delta_{i-1} - \alpha \frac{\partial L}{\partial \delta^{i-1}}$
    Compute $\hat{z}_{\delta_i}^{t+1} = z^t + (W_1 \delta_i) \odot \mathbf{a} \begin{bmatrix} z^t & z^{t-1} \end{bmatrix}^T$
    Compute $\hat{x}_i^{t+1} = G_W(z_{\delta^i}^{t+1})$
  end for
  Back propagate the full encoder/predictor loss from Equation 7 using δ_k, and update the weight matrices W and W_1
end for

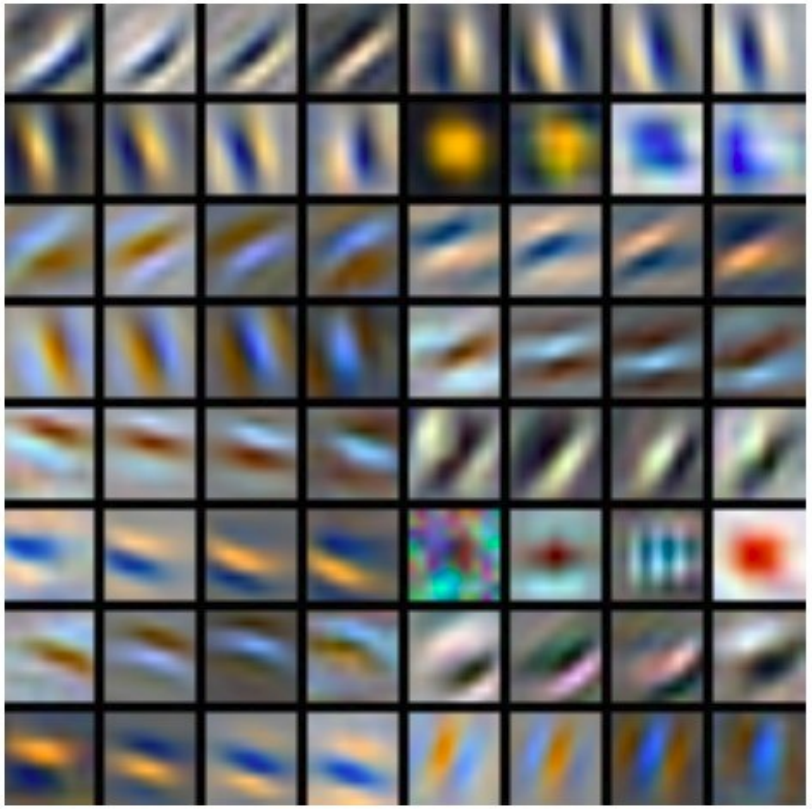
$$\hat{z}_\delta^{t+1} = z^t + (W_1 \delta) \odot \mathbf{a} \begin{bmatrix} z^t & z^{t-1} \end{bmatrix}^T$$

$$L = \min_\delta \|G_W(\hat{z}_\delta^{t+1}) - x^{t+1}\|_2^2 - \lambda \frac{(z^t - z^{t-1})^T (z^{t+1} - z^t)}{\|z^t - z^{t-1}\| \|z^{t+1} - z^t\|}$$

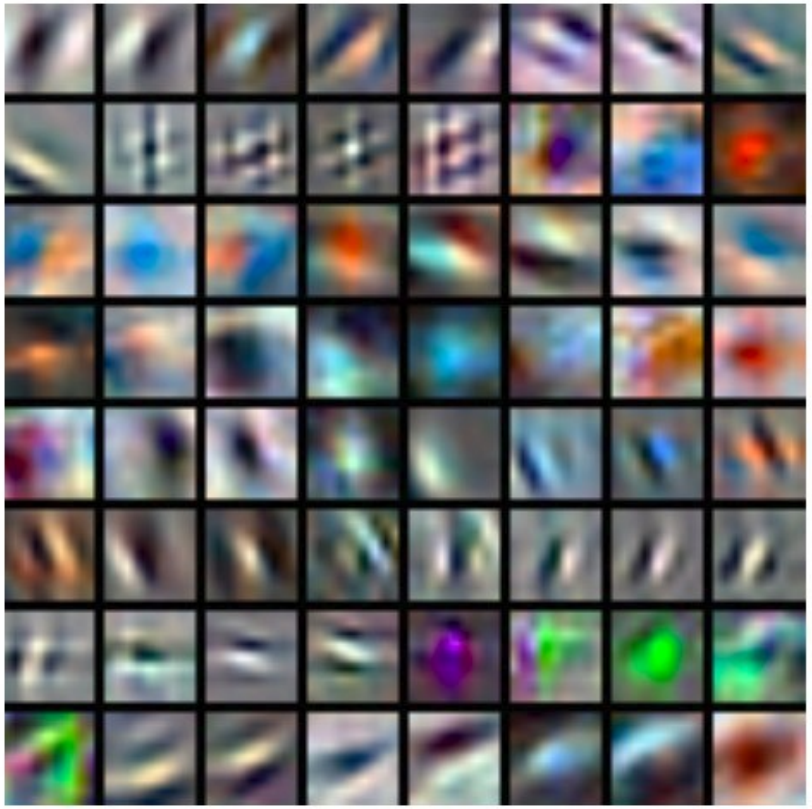
Architectures

	Encoder	Prediction	Decoder
Shallow Architecture 1	Conv+ReLU $64 \times 9 \times 9$ Phase Pool 4	Average Mag. Linear Extrap. Phase	Conv $64 \times 9 \times 9$
Shallow Architecture 2	Conv+ReLU $64 \times 9 \times 9$ Phase Pool 4 stride 2	Average Mag. Linear Extrap. Phase	Conv $64 \times 9 \times 9$
Deep Architecture 1	Conv+ReLU $16 \times 9 \times 9$ Conv+ReLU $32 \times 9 \times 9$ FC+ReLU 8192×4096	None	FC+ReLU $\mathbf{8192} \times 8192$ Reshape $32 \times 16 \times 16$ SpatialPadding 8×8 Conv+ReLU $16 \times 9 \times 9$ SpatialPadding 8×8 Conv $1 \times 9 \times 9$
Deep Architecture 2	Conv+ReLU $16 \times 9 \times 9$ Conv+ReLU $32 \times 9 \times 9$ FC+ReLU 8192×4096	Linear Extrapolation	FC+ReLU 4096×8192 Reshape $32 \times 16 \times 16$ SpatialPadding 8×8 Conv+ReLU $16 \times 9 \times 9$ SpatialPadding 8×8 Conv $1 \times 9 \times 9$
Deep Architecture 3	Conv+ReLU $16 \times 9 \times 9$ Conv+ReLU $32 \times 9 \times 9$ FC+ReLU 8192×4096 Reshape $64 \times 8 \times 8$ Phase Pool 8×8	Average Mag. Linear Extrap. Phase	Unpool 8×8 FC+ReLU 4096×8192 Reshape $32 \times 16 \times 16$ SpatialPadding 8×8 Conv+ReLU $16 \times 9 \times 9$ SpatialPadding 8×8 Conv $1 \times 9 \times 9$

Filters: pooling over groups of 4 filters.



(a) Shallow Architecture 1



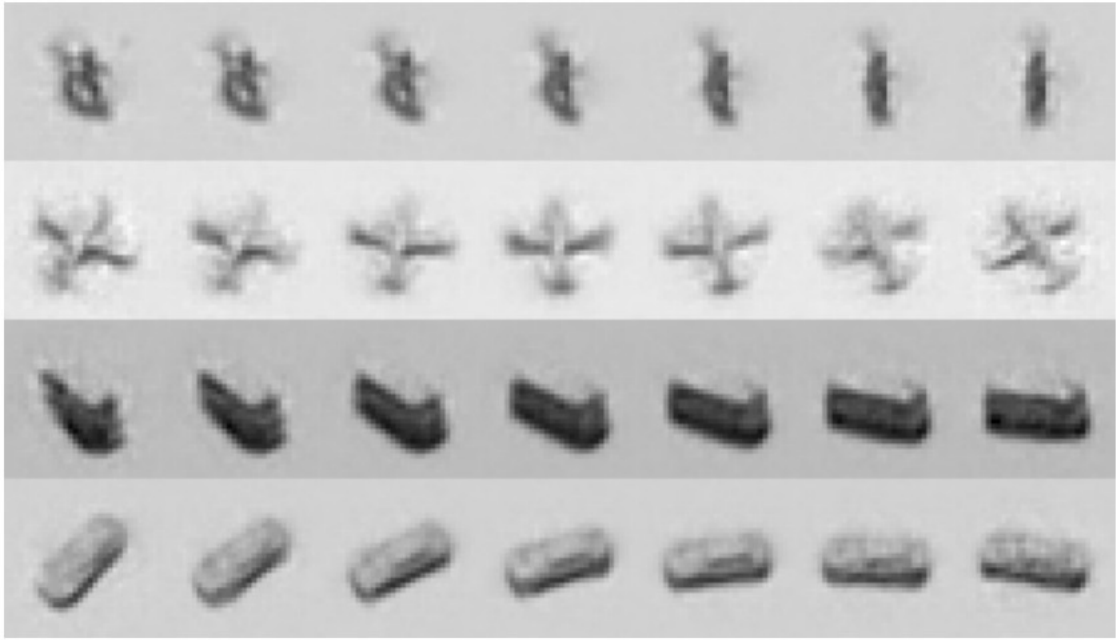
(b) Shallow Architecture 2

Non-overlapping pooling
4x4x4 (feat,x,y)

Overlapping pooling
4x4x4 (feat,x,y)
Stride 2 over features

Image interpolation in feature space: deterministic world

- No phase pooling
- No curvature regularization



- With phase pooling
- With curvature regularization

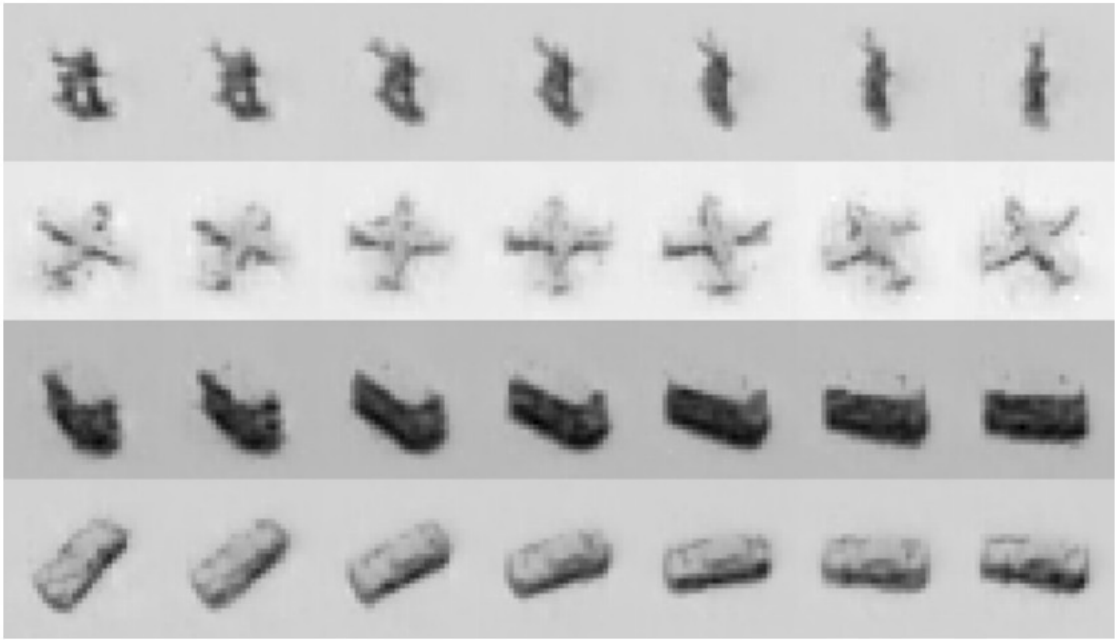


Image interpolation in feature space: deterministic world

- Image interpolation with the basic model (siamese net)

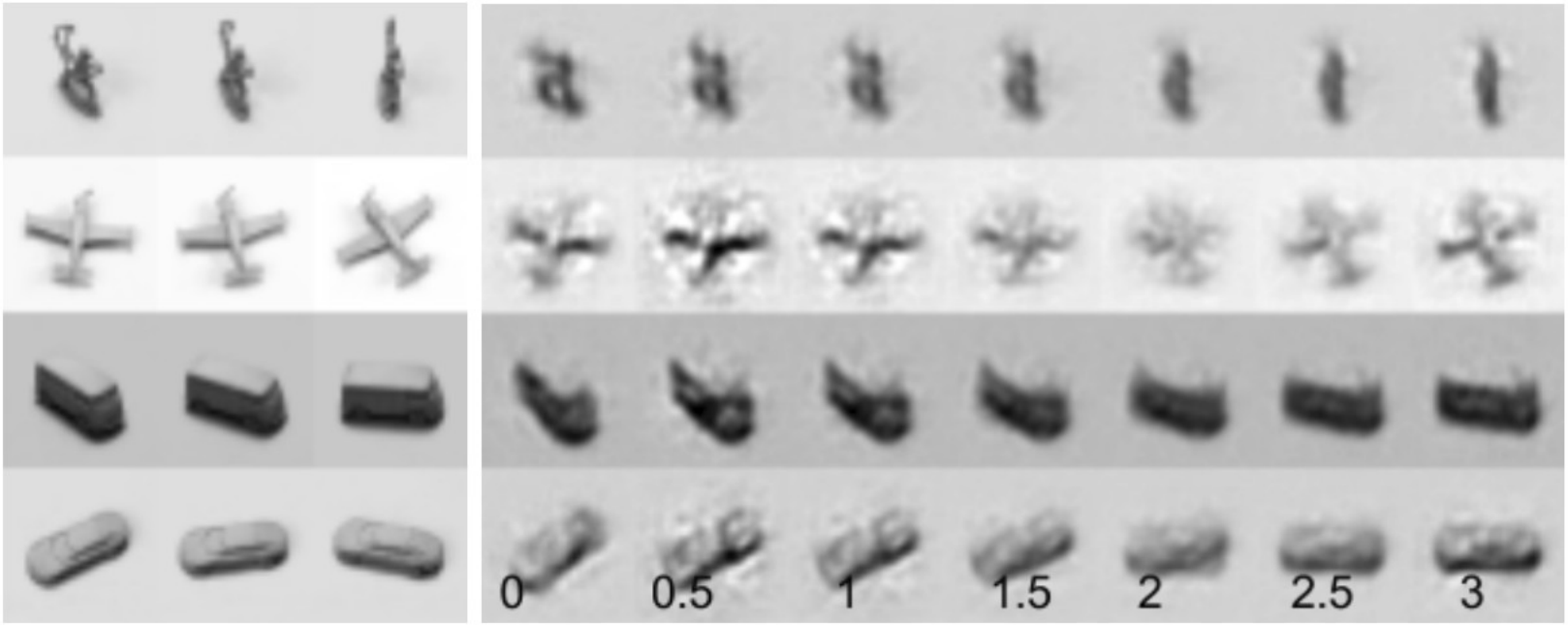
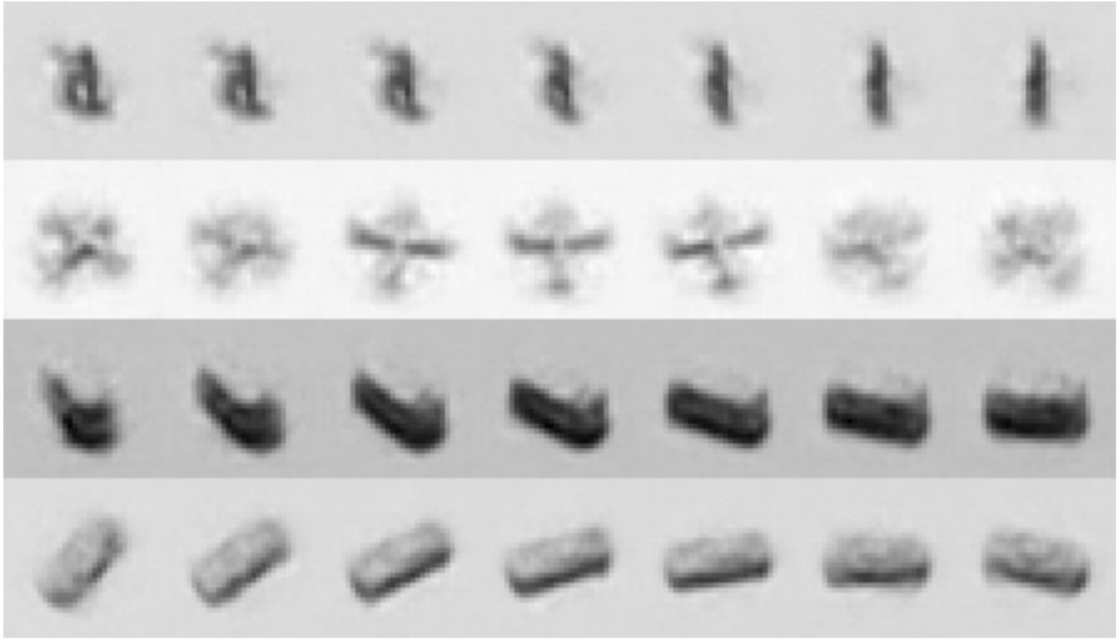
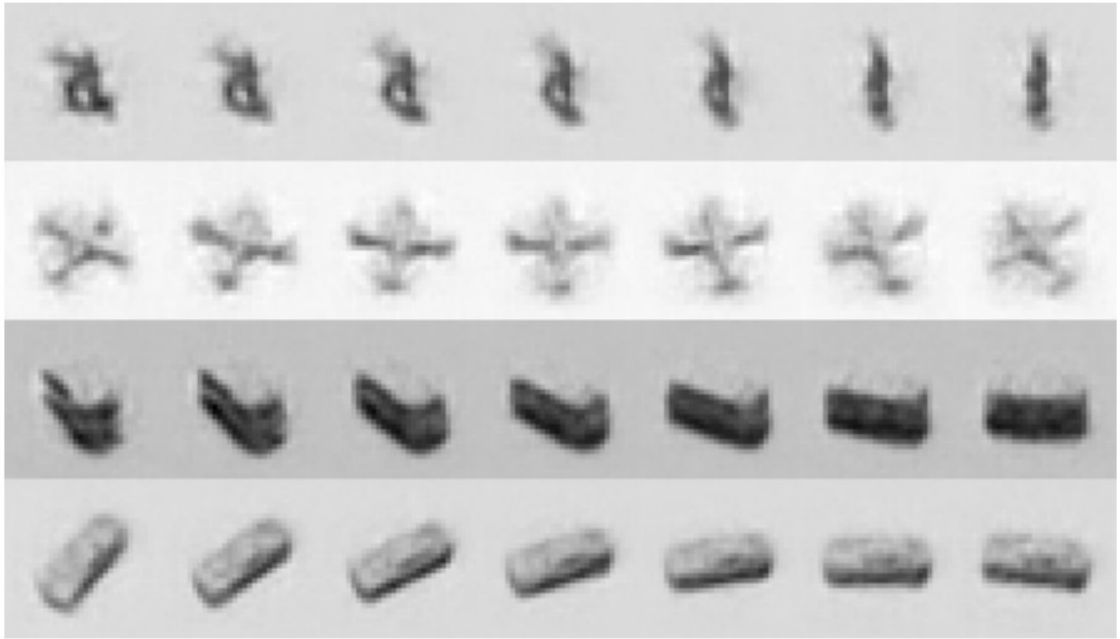


Image interpolation in feature space: unpredictable world

- Phase interpolation
- No latent variable



- Phase interpolation
- With latent variable

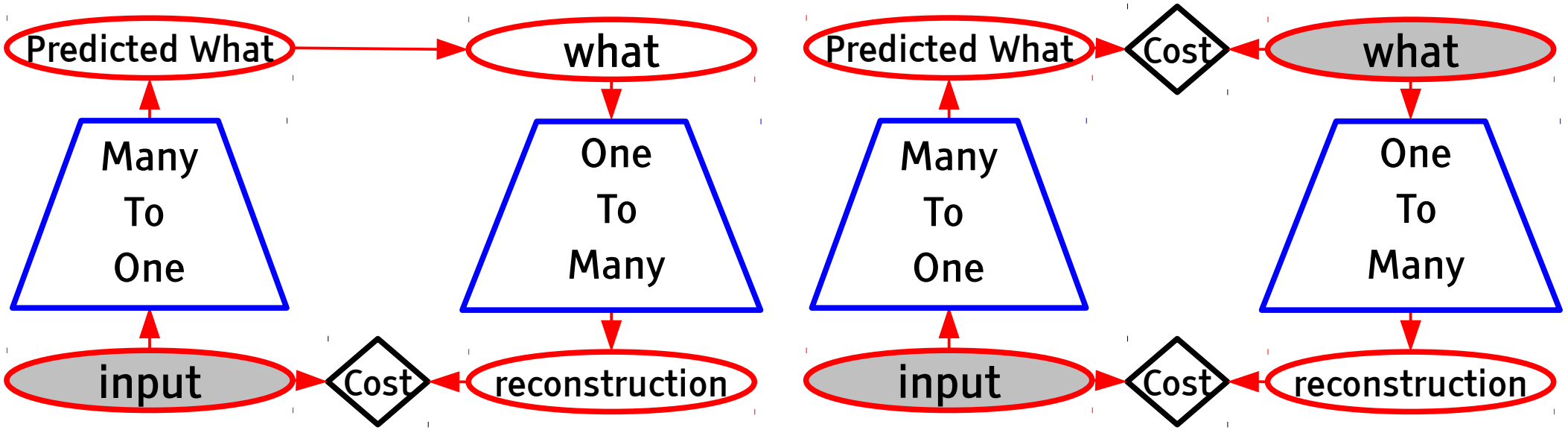


Integrating Supervised & Unsupervised Learning With Stacked What-Where Auto-Encoders

Supervised and Unsupervised in One Learning Rule?

- **Boltzmann Machines have all the right properties** **[Hinton 1893] [OK, OK 1983 ;-]**
  - Sup & unsup, generative & discriminative in one simple/local learning rule
  - Feedback circuit reconstructs and propagates virtual hidden targets
  - But they don't really work (or at least they don't scale).
- **Problem: the feedforward path eliminates information**
- **If the feedforward path is invariant, then**

  the reconstruction path is a one-to-many mapping
  - Usual solution: sampling. But I'm allergic.



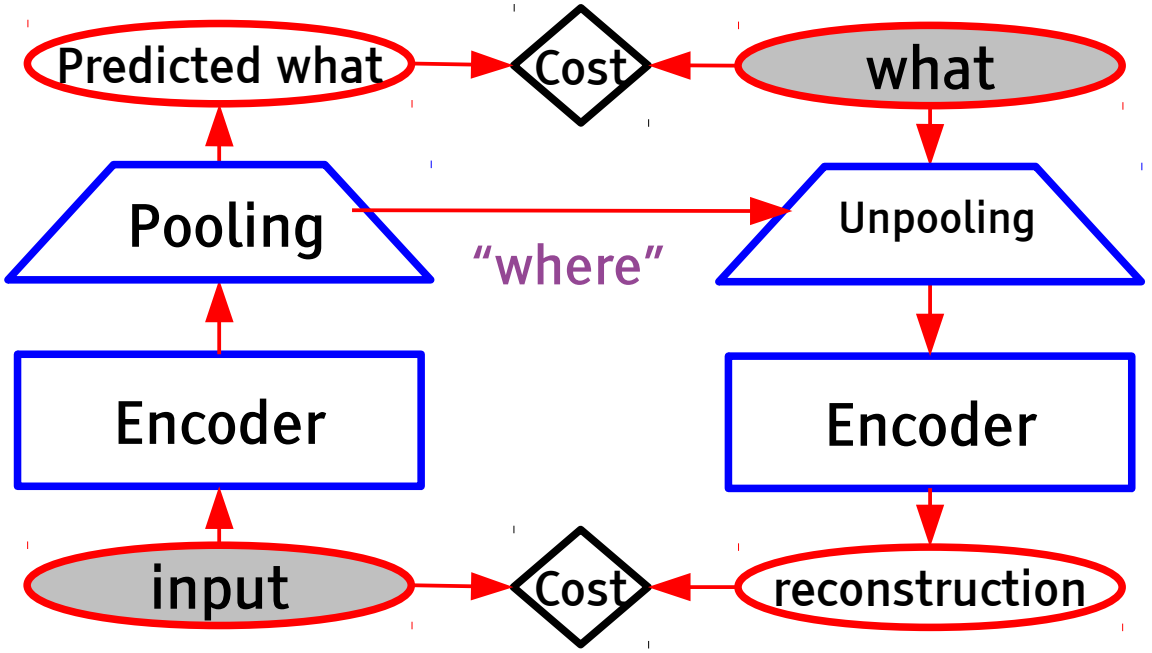
Supervised and Unsupervised in One Learning Rule?

Idea: keep the complementary information around

- So that the generative path is a function

What-Where Auto-Encoder

- [Ranzato 2007], [Gregor 2011], Hinton's Capsules
- Very old ideas by Hinton & Zemel, von der Malsburg and others about separating identity from instantiation parameters.

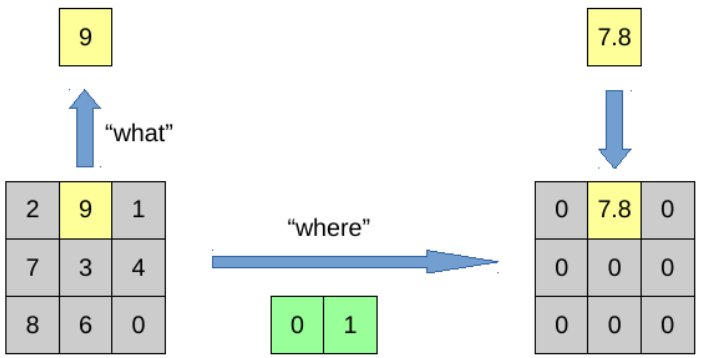


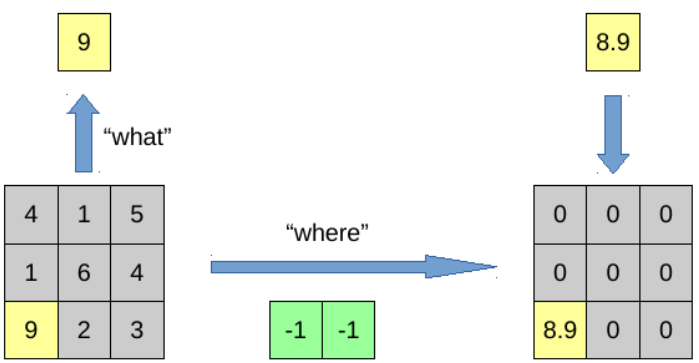
Computing the "where": Phase Pooling

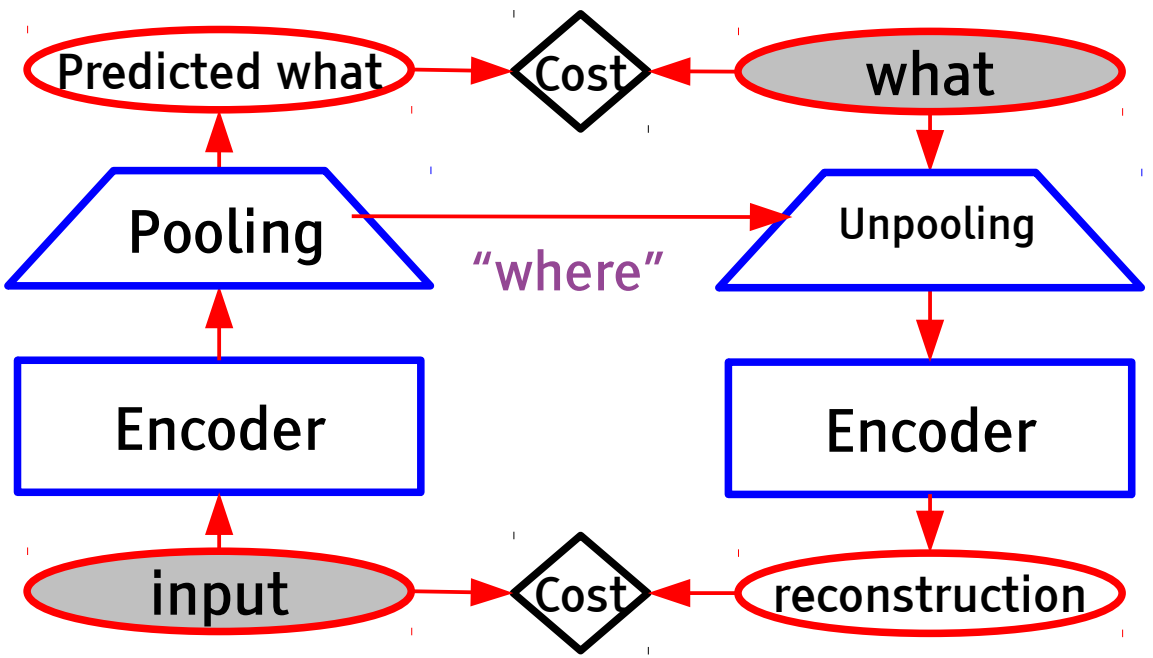
A funny kind of pooling/unpooling

$$m_k = \sum_{N_k} z(f,x,y) \frac{e^{\beta z(f,x,y)}}{\sum_{N_k} e^{\beta z(f',x',y')}} \approx \max_{N_k} z(f,x,y)$$

$$\mathbf{p}_k = \sum_{N_k} \begin{bmatrix} f \\ x \\ y \end{bmatrix} \frac{e^{\beta z(f,x,y)}}{\sum_{N_k} e^{\beta z(f',x',y')}} \approx \arg\max_{N_k} z(f,x,y)$$



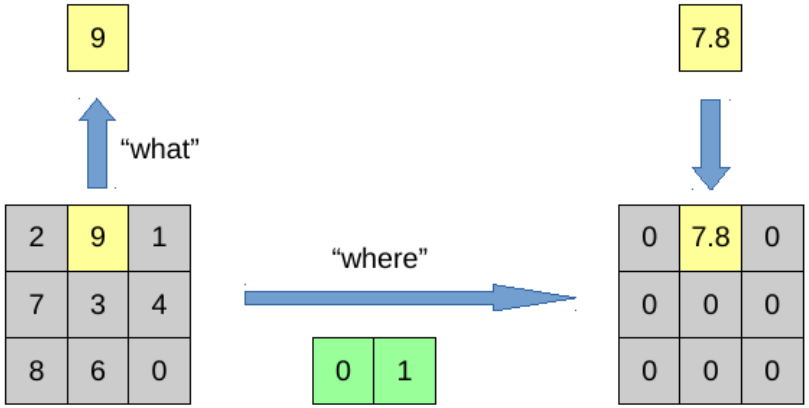




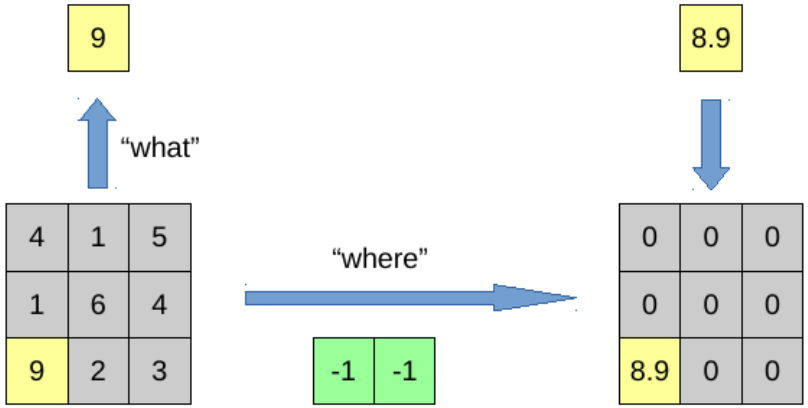
Stacked What-Where Auto-Encoder (SWWAE)

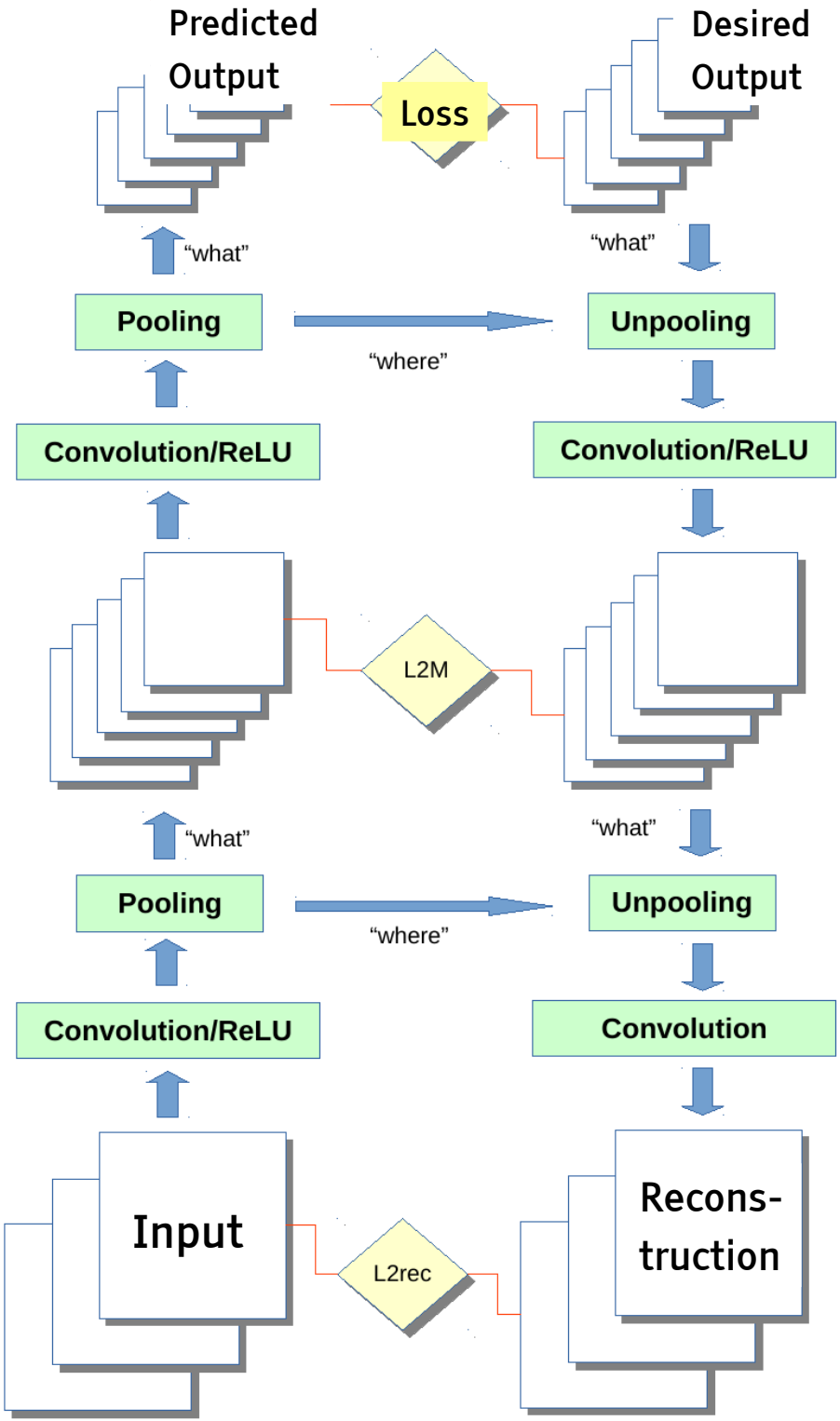
[Zhao, Mathieu, LeCun arXiv:1506.02351]

Stacked What-Where Auto-Encoder



A bit like a ConvNet paired with a DeConvNet





SWWAE: Reconstructions with Unpooling

Network: MNNIST → [Conv 5x5] → 16 fmaps → Conv 3x3 → 32 fmaps → Pooling PxP

Trained unsupervised. Hard max-pooling at test time.

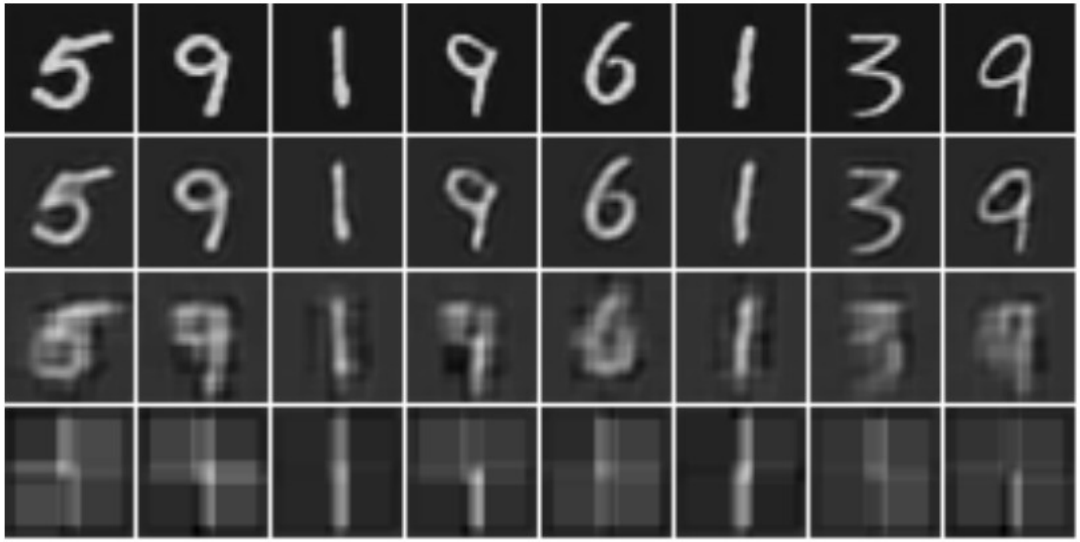
Upsampling

P=2

P=4

P=8

P=16



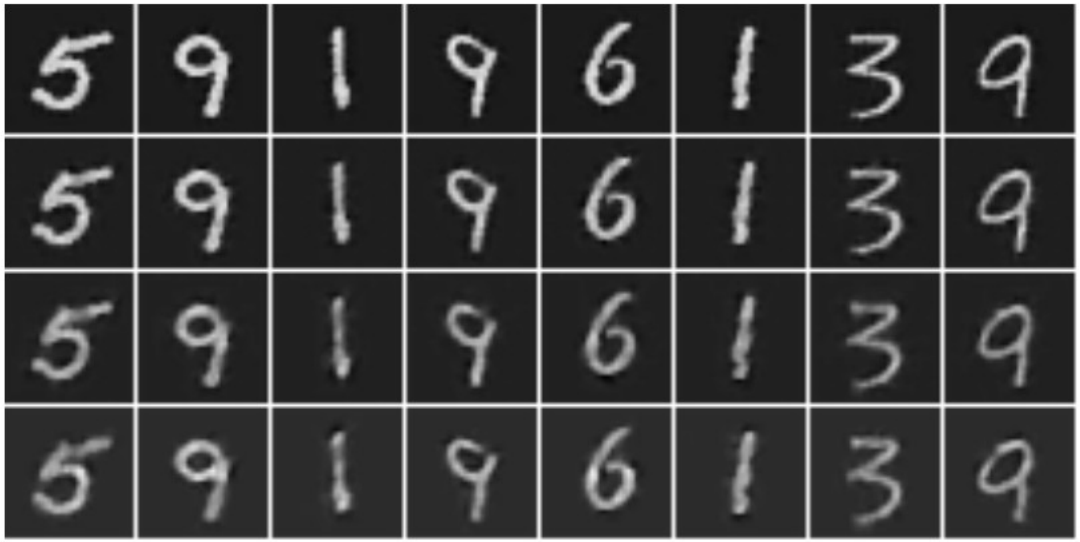
Unpooling

P=2

P=4

P=8

P=16

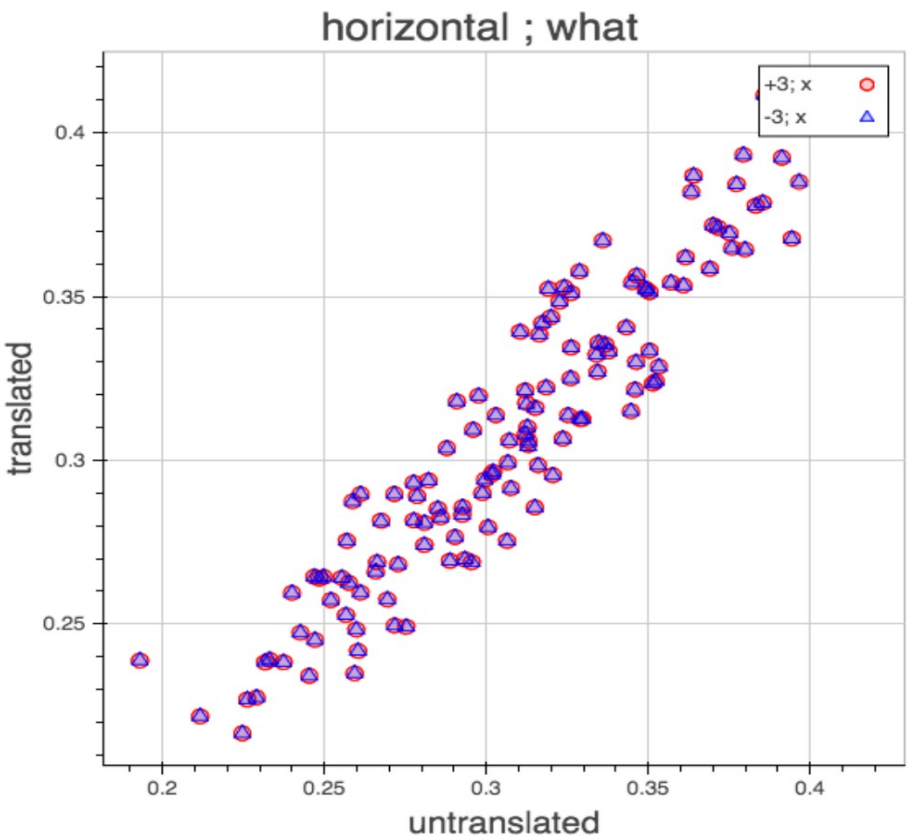


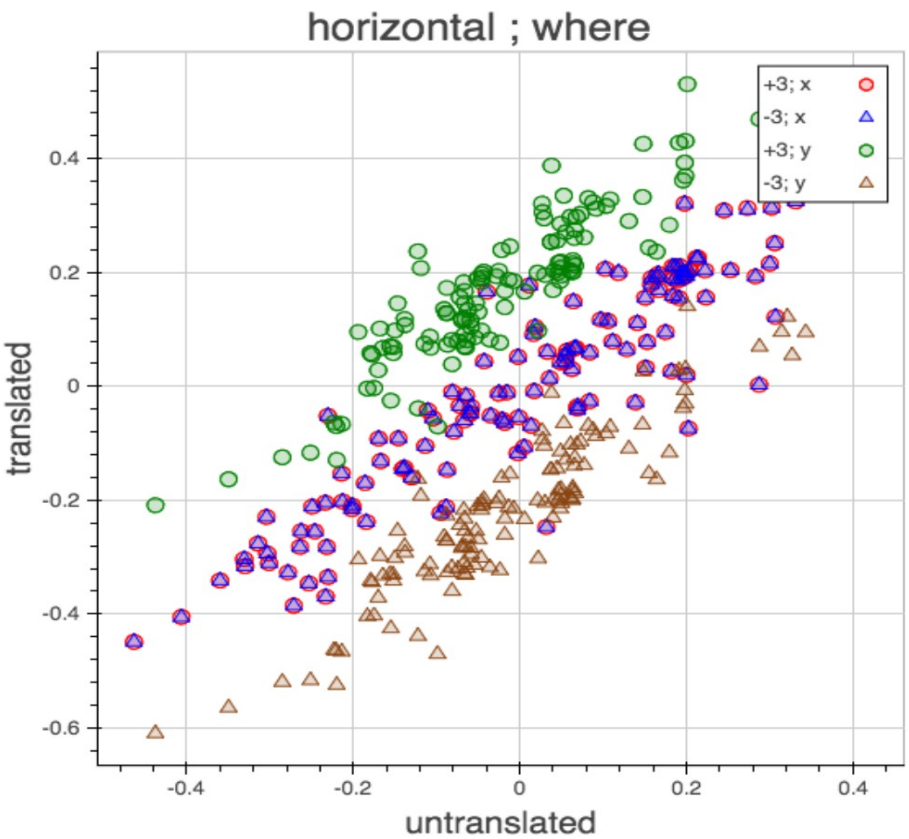
SWWAE: “What” is invariant to translations. “Where” isn't

Activations of features for untranslated and translated inputs

Left: “what”. Activations hardly change with -3 and +3 translations

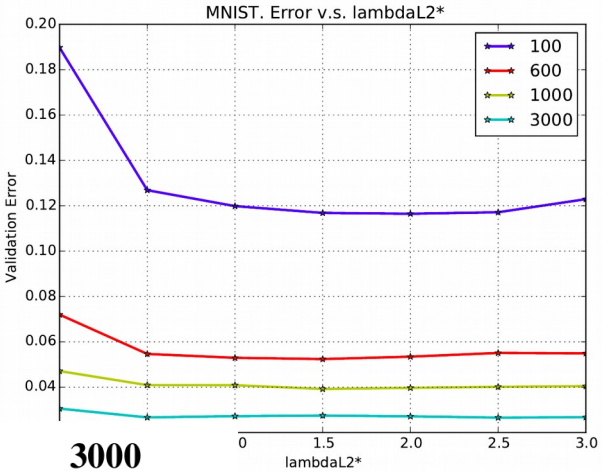
Right: “Where”. Activity changes with shift.





MNIST: Recognition & Generation

model / N	100	600	1000	3000
SWWAE	[1]11.65 ± 0.20	[1]5.24 ± 0.06	[1]3.92 ± 0.09	[2]2.66 ± 0.07
dp	21.11 ± 0.65	7.11 ± 0.27	5.34 ± 0.39	3.23 ± 0.30
dp-fc	16.09 ± 0.96	6.14 ± 0.36	4.368 ± 0.50	2.98 ± 0.08
unsup-sfx	17.81 ± 0.06	8.41 ± 0.08	6.40 ± 0.06	4.76 ± 0.03
unsup-pretr	-	9.80 ± 0.06	6.135 ± 0.03	4.41 ± 3.11
noL2M	[2]13.48 ± 2.35	[2]5.69 ± 0.33	[2]3.97 ± 0.37	[1]2.52 ± 0.06



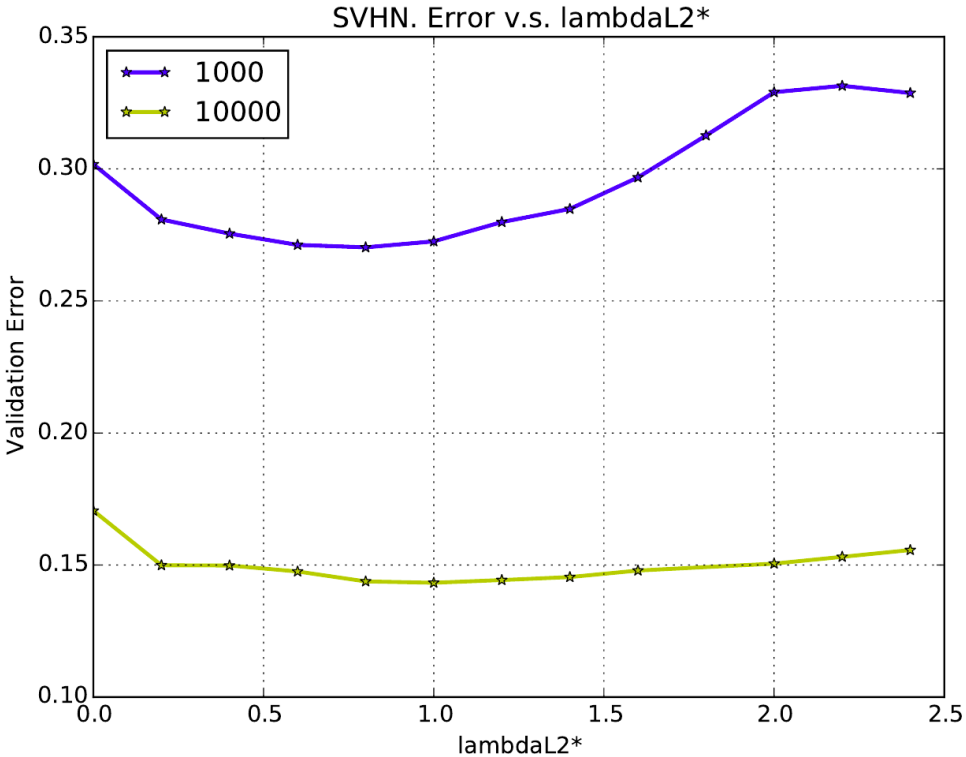
model / N	100	600	1000	3000
Convnet (LeCun et al. (1998))	22.98	7.86	6.45	3.35
TSVM (Vapnik & Vapnik (1998))	16.81	6.16	5.38	3.45
CAE (Rifai et al. (2011b))	13.47	6.3	4.77	3.22
MTC (Rifai et al. (2011a))	12.03	5.13	3.64	2.57
PL-DAE (Lee (2013))	10.49	5.03	3.46	2.69
WTA-AE (Makhzani & Frey (2014))	-	**2.37**	1.92	-
M1+M2 (Kingma et al. (2014))	3.33 ± 0.14	2.59 ± 0.05	2.40 ± 0.02	2.18 ± 0.04
LadderNetwork (Rasmus et al. (2015a))	$\mathbf{1.13 \pm 0.04}$	-	$\mathbf{1.00 \pm 0.06}$	-
SWWAE without dropout	9.17 ± 0.11	4.16 ± 0.11	3.39 ± 0.01	2.50 ± 0.01
SWWAE with dropout	8.71 ± 0.34	3.31 ± 0.40	2.83 ± 0.10	$\mathbf{2.10 \pm 0.22}$

SVHN: Recognition

- House Numbers
- Network:

(128)5c-2p-(128)3c-(256)3c-2p-(256)3c-2p

- Error rate on full dataset
  - No reconstruction: 5.89%
  - With reconstruction: 4.94%



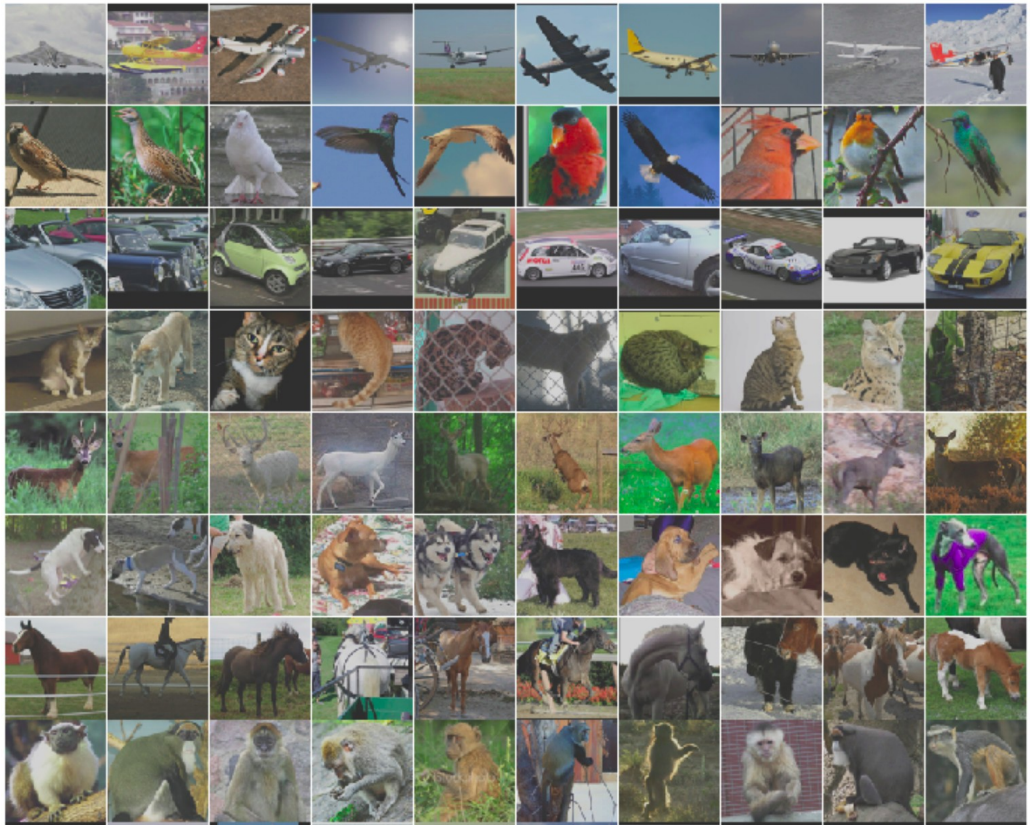
model / N	1000	10000
SWWAE	[1]27.03	[1]14.33
dp	32.25	15.72
dp-fc	[2]27.81	[2]14.39
L1	30.22	14.57
unsup-pretr	33.03	17.31
noL2M	29.12	16.51

model / N	1000 labels	error rate (in %)
KNN		77.93
TSVM		66.55
M1+KNN (Kingma et al. (2014))		65.63
M1+TSVM (Kingma et al. (2014))		54.33
M1+M2 (Kingma et al. (2014))		36.02
SWWAE without dropout ($\lambda_{L2*} = 0.8$)		27.83
SWWAE with dropout ($\lambda_{L2*} = 0.4$)		**23.56**

STL-10: Recognition

- 10 classes: airplane, bird, cat, car, deer, dog, horse, monkey, ship, truck
- 96x96 color images (from ImageNet)
- 10 predefined folds with 1000 training samples each (100 per class)
- 5000 total training samples
- 100K unlabeled samples (other categories)
- 8000 test samples (800 per class)

  [Coates et al. 2011]
- (64)3c-4p-(64)3c-3p-(128)3c-(128)3c-2p-(256)3c-(256)3c-(256)3c-(512)3c-(512)3c-(512)3c-2p-10fc

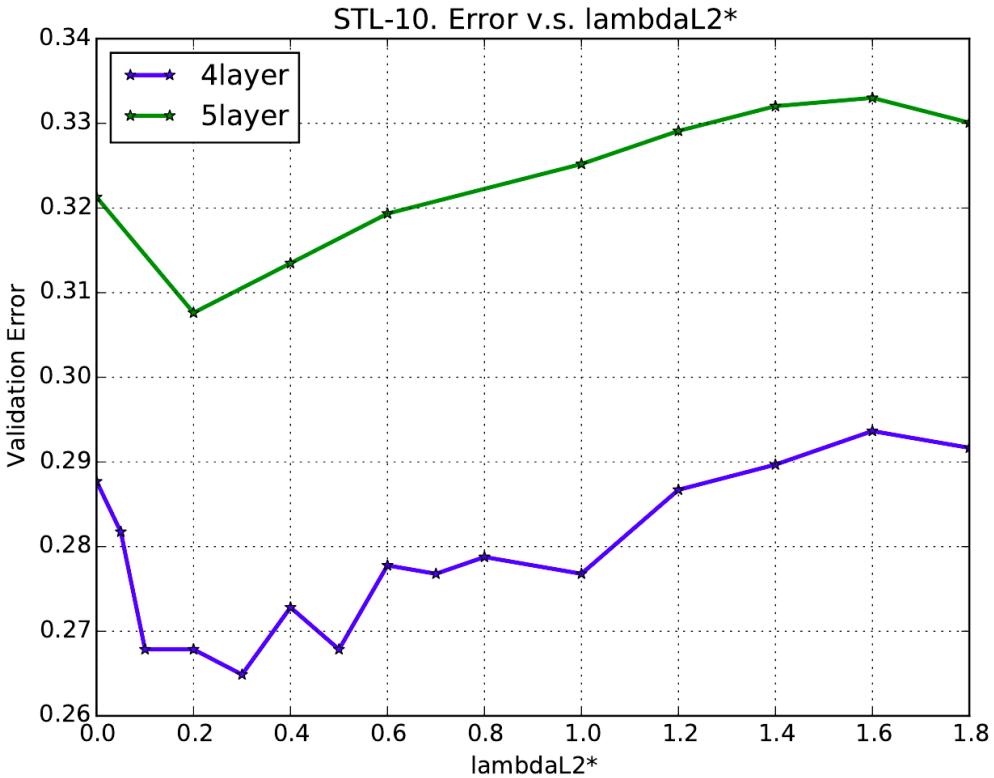


model	accuracy
Multi-task Bayesian Optimization (Swersky et al. (2013))	70.1%
Zero-bias Convnets + ADCU (Paine et al. (2014))	70.2%
Exemplar Convnets (Dosovitskiy et al. (2014))	**75.4%**
SWWAE	74.33%
Convnet of same configuration	57.45%

STL-10: Recognition

- 10 classes: airplane, bird, cat, car, deer, dog, horse, monkey, ship, truck
- 96x96 color images (from ImageNet)
- 10 predefined folds with 1000 training samples each (100 per class)
- 5000 total training samples
- 100K unlabeled samples (other categories)
- 8000 test samples (800 per class)

  [Coates et al. 2011]



model	accuracy(val)
SWWAE-4layer	[1]73.51%
SWWAE-5layer	69.24%
noL2M-5layer	68.26%
noL2M-4layer	70.93%
dp-4layer	70.54%
dp-fc-4layer	71.43%

model	accuracy
Convolutional Kernel Networks [17]	62.32%
HMP [1]	64.5%
NOMP [16]	67.9%
Multi-task Bayesian Optimization [27]	70.1%
Zero-bias Convnets + ADCU [20]	70.2%
Exemplar Convnets [2]	72.8%
SWWAE-4layer	[1]74.80%

CIFAR-10, CIFAR-100, Unsup on 80M Tiny Images

- 60,000 labeled samples
- 80 Million unlabeled samples
- (128)3c-(256)3c-2p-(256)3c-(512)3c-2p-(512)3c-(512)3c-2p-(512)3c-(512)3c-2p-128fc-10fc

model	CIFAR-10	CIFAR-100
All-Convnet (Springenberg et al. (2014))	**92.75%**	66.29%
Highway Network (Srivastava et al. (2015))	92.40%	67.76%
Deeply-supervised nets (Lee et al. (2014))	92.03%	65.43%
Fractional Max-pooling with large augmentation (Graham (2014))	95.50%	68.55%
SWWAE ($\lambda_{L2rec} = 1, \lambda_{L2M} = 0.2$)	92.23%	**69.12%**
Convnet of same configuration	91.33%	67.50%